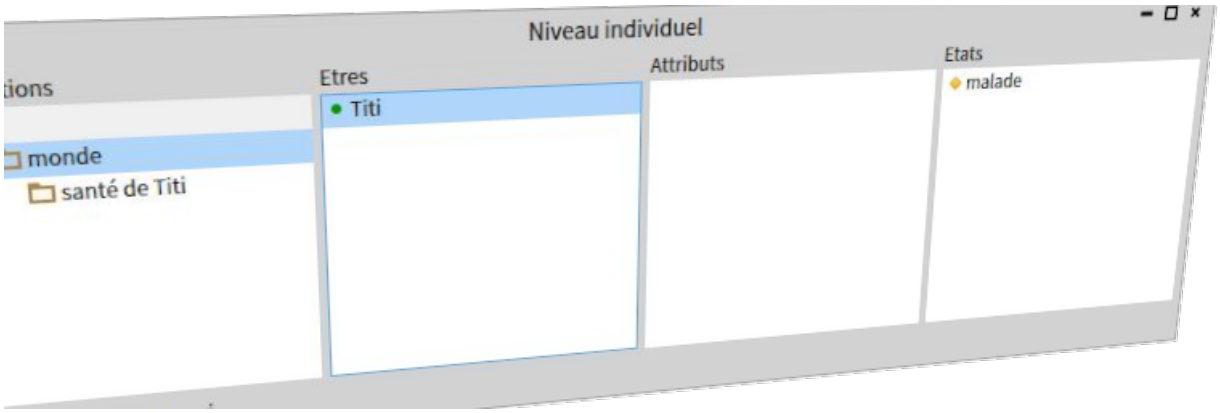
Présentation du langage d'ICEO et de son implantation en Smalltalk (Pharo)

ICEO par l'exemple

Robert Bourgeois



robert.bourgeois@ieee.org

06/05/2024

Ce document a été écrit en utilisant l'application TeXstudio 4.7.3

Les graphes présentés dans ce document ont été réalisés avec l'outil graphique Yed de yWorks

Contents

Introduction

Les concepts illustrés ici par des exemples sont présentés dans le document intitulé "Sur la modélisation des êtres et de leurs manières d'être dans certaines situations".

Ce document est téléchargeable à l'adresse https://github.com/rodejaphgh/ICEO.

Le langage linéaire d'ICEO est implanté en Smalltalk[1]

Le choix de ce langage est lié à l'élégance de sa syntaxe mais aussi et surtout au fait que de nombreux concepts d'ICEO sont inspirés ou sont ceux de ce langage. La preuve en est que l'implantation d'ICEO en Smalltalk se contente de quelques centaines de lignes de code.

Nous avons utilisé Pharo qui est, parmi les versions de Smalltalk dérivées de la version de Smalltalk-80 originelle, l'une des plus proches.

Cette construction n'a quasiment exigé aucune modification de l'environnement de base de Pharo.

Pour étudier ces exemples, nous vous proposons de le faire de manière interactive dans l'environnement intégré de Pharo.

Ceci sera peut-être pour certains l'occasion de découvrir ce merveilleux langage qu'est Smalltalk.

[1]Smalltalk est un langage de programmation orienté objet qui a été conçu en 1972 par des génies informatiques : Alan Kay, Dan Ingals, Ted Kaehler et Adele Goldberg au Palo Alto de Xerox. Ce langage qui a inspiré une multitude d'autres langages (tels que Java, C++, Python, ...) fut l'un des tout premiers à disposer d'un environnement de développement intégré complètement graphique.

Installation d'ICEO dans Pharo et exemples

Pharo est disponible librement en téléchargement sur le site https://pharo.org/download

Pour nos lecteurs qui ne connaissent pas Smalltalk, en particulier dans sa version Pharo, nous conseillons de commencer par la lecture du livre "*Pharo By Example*" de Stéphane Ducasse, Gordana Rakic, Sebastian Kaplar et Quentin Ducasse dont une version libre au format pdf peut être téléchargée à l'adresse

https://books.pharo.org/pharo-by-example9

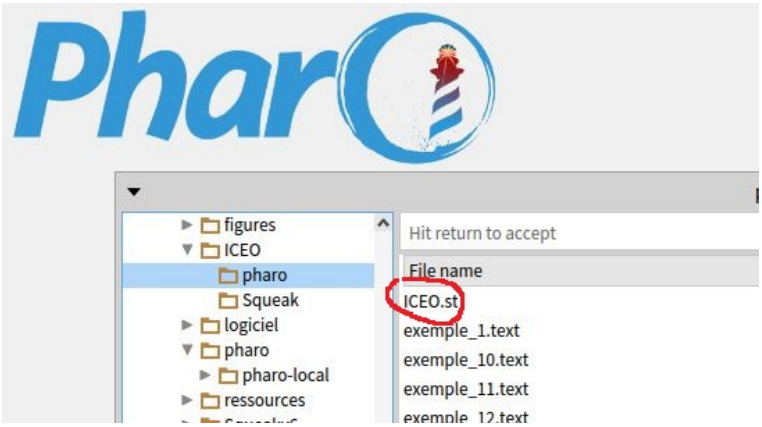
Installation d'ICEO

Les fichiers sources d'ICEO sont téléchargeables à l'adresse https://github.com/rodejaphgh/ICEO

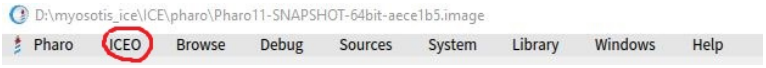
ou à l'adresse alternative isnel.net/ICEO.zip

Ils comprennent le code source d'ICEO (fichier ICEO.st) et le code des exemples présentés dans ce document.

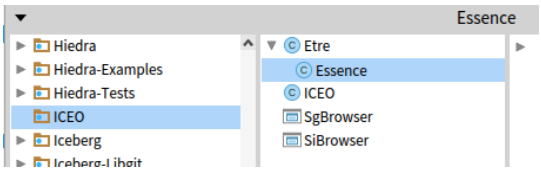
Pour installer ICEO dans l'environnement de Pharo, il convient d'utiliser l'outil "File Browser" accessible sous l'onglet "System" dans la barre d'outils de Pharo. Exécuter l'option "Install into image" du fichier ICEO.st précédemment téléchargé :



Attendre quelques instants (soyez patient !) que la barre de menu principale affiche un nouvel onglet nommé ICEO :



Dans un Browser, la catégorie ICEO s'affiche avec les classes Etre, Essence, et ICEO :

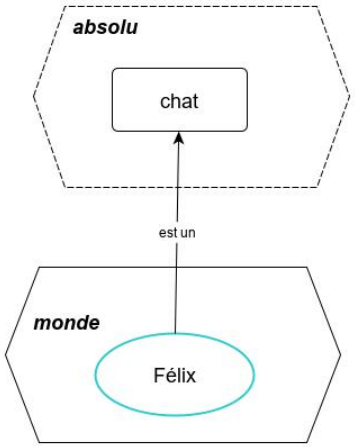


La classe ICEO est l'interprète du langage linéaire d'ICEO.

Une instance de cette classe est créée lors de l'installation, nommée "iceo" (en minuscules).

Avant de continuer, il est conseillé de sauvegarder votre image. Pour retrouver Pharo dans l'état actuel lors du prochain lancement, il suffira ainsi de choisir cette image.

Intéressons-nous maintenant au premier exemple, qui correspond au graphe suivant :



Dans la fenêtre File Browser sélectionner le fichier "exemple_1.text"

Le texte suivant s'affiche :

```
1 iceo definition: #chat.
2 iceo soit: #Félix essence: chat.
```

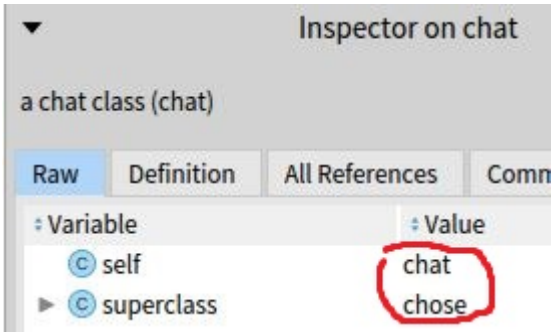
Rappelons que dans ICEO le nom commun des essences s'écrit obligatoirement (comme en langage naturel) avec une minuscule et que le nom propre des êtres s'écrit habituellement avec une majuscule.

Dans ce texte, vous pouvez interpréter les lignes une par une ou en bloc, en utilisant l'option "Do it" offerte avec le bouton droit de la souris. Pour l'interprétation d'un bloc de lignes, il faut que chaque ligne se termine par un point. Chaque exemple peut être exécuté en une seule passe en sélectionnant l'ensemble des lignes par la commade CTRL A suivie de la commande CTRL D.

Un conseil : ayez toujours une fenêtre de type Transcript ouverte lors de l'exécution d'un bloc de lignes. En cas d'erreur, la ligne fautive y sera affichée.

La première ligne de l'exemple définit l'essence chat dans la situation générique absolu, subsumée par l'essence chose.

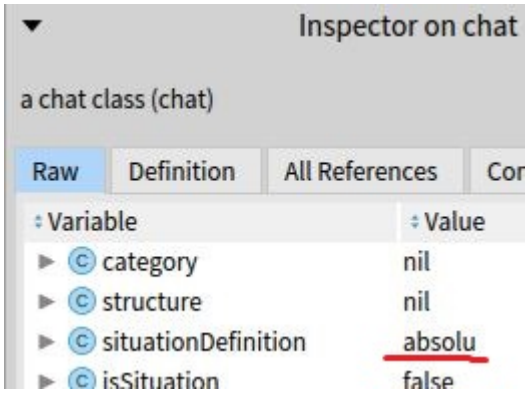
Pour le vérifier, inspecter le mot chat (utiliser l'option "Inspect it" offerte avec le bouton droit de la souris) :



On voit que l'essence chat a été créée sous la forme d'une classe Smalltalk. Les différents attributs qui sont listés seront expliqués au moment opportun.

L'attribut "superclass" montre que le genus de l'essence chat est l'essence chose.

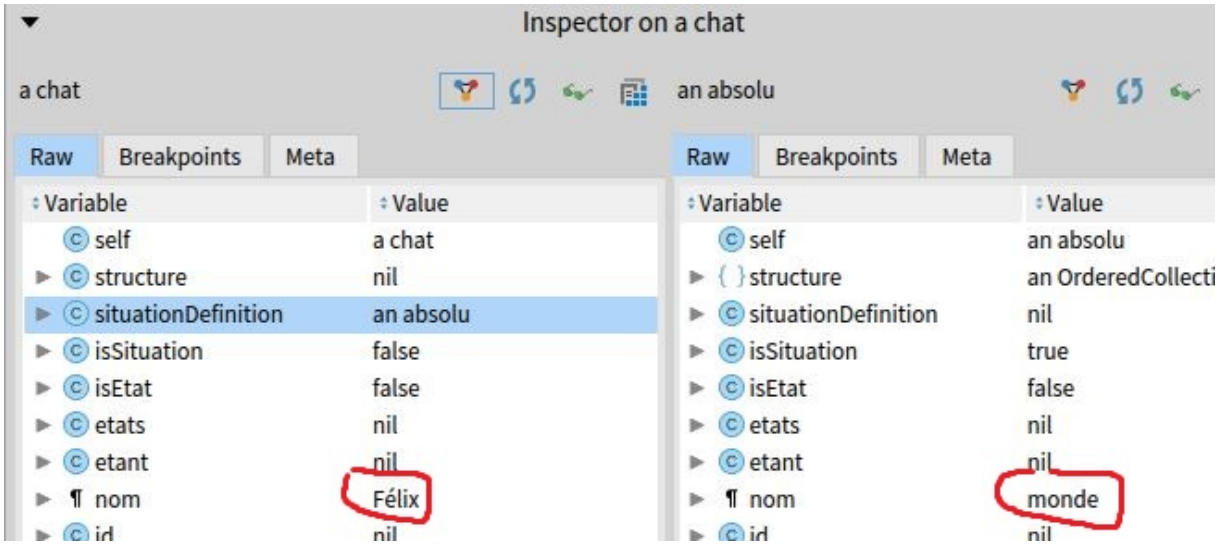
L'attribut "situationDefinition" montre que l'essence chat est définie dans l'absolu :



L'accès à l'essence chat pourrait se faire avec l'expression " `absolu get: #chat` " mais ce n'est pas nécessaire car les essences définies dans l'absolu sont accessibles directement par leur nom.

La deuxième ligne crée l'être nommé Félix comme instance de chat dans le monde.

L'expression "`monde get: #Félix` " montre que Félix est un chat présent dans le monde :

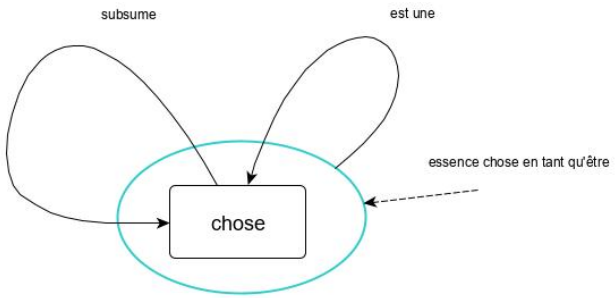


Noter que l'accès à l'être nommé Félix situé dans le monde exige de passer par l'expression "`monde get: #Félix` ", car Félix ne se situe pas dans l'absolu comme l'essence chat mais dans le monde.

L'essence chose définie dans l'absolu est son propre genus et, en tant qu'être, est instance d'elle-même (elle est sa propre méta-essence).

Ainsi les expressions " `chose getGenus` " et " `chose getEssence` " retournent chose.

Ceci est conforme au schéma :



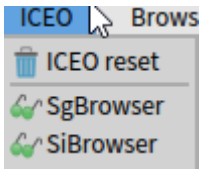
Par défaut, l'essence d'une essence (sa méta-essence) est l'essence chose.

Ainsi, l'expression " chat getEssence " retourne chose.

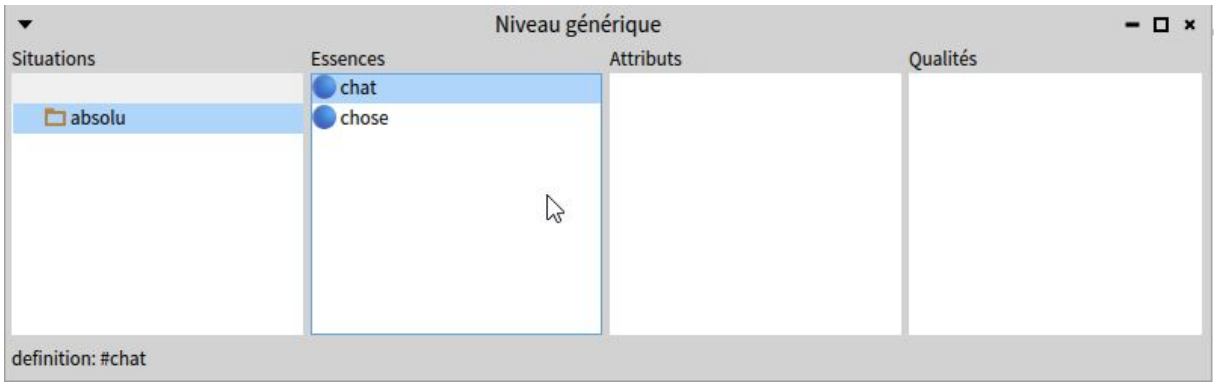
Vous êtes maintenant parés pour étudier d'autres exemples.

Services accessibles via l'onglet "ICEO" de la fenêtre principale

Ces services sont également accessibles avec le bouton gauche de la souris dans la fenêtre principale.



L'option "SgBrowser" ouvre une fenêtre sur les situations génériques et les essences définies dans l'absolu

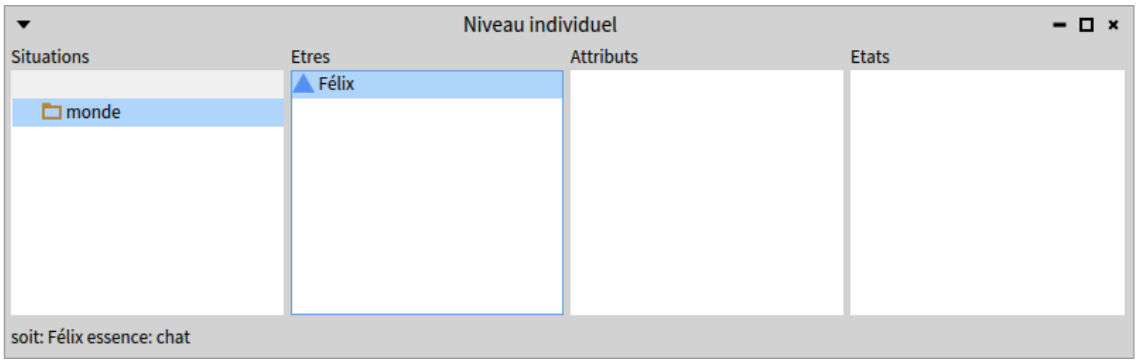


Le nom des essences est précédé d'un petit cercle bleu.

Un double-click sur un item (par exemple l'essence chat) ouvre une fenêtre permettant d'inspecter celle-ci.

Le label affiché en bas de la fenêtre rappelle l'expression qui a donné naissance à l'entité sélectionnée.

L'option "SiBrowser" ouvre une fenêtre sur les situations individuelles et les êtres définis dans le monde :



Le nom des êtres est précédé d'un petit triangle bleu.

L'option "ICEO reset" réinitialise le contenu de absolu et de monde. Ceci évite qu'ICEO ne conserve la mémoire des exemples précédemment étudiés.

En principe, cette instruction n'est pas nécessaire ici, car il s'agit de notre premier exemple.

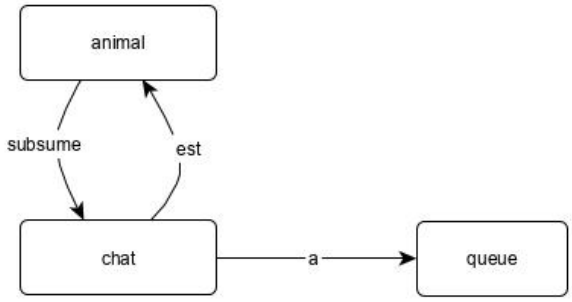
Par contre, il pourra être nécessaire de l'exécuter avant de démarrer l'étude d'un autre exemple, pour éviter des interférences avec le précédent.

Rappelons que dans ICEO :

- deux essences de même nom ne peuvent être définies dans la même situation.
- deux êtres de même nom ne peuvent coexister dans une situation, sauf s'ils peuvent être distingués par l'un de leurs états (par exemple, deux êtres nommés Pierre dont l'un est fils et l'autre père dans une même famille).

Exemple 2 : sur la composition d'une essence

Cet exemple correspond au graphe suivant :



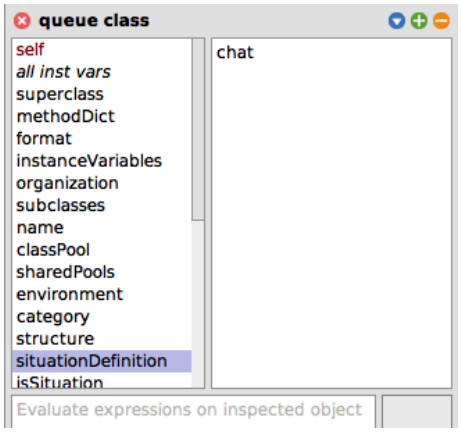
chat est animal qui a queue (chat = ⊕ (animal, { queue}))

L'essence chat subsumée par animal possède un attribut propre nommé queue :

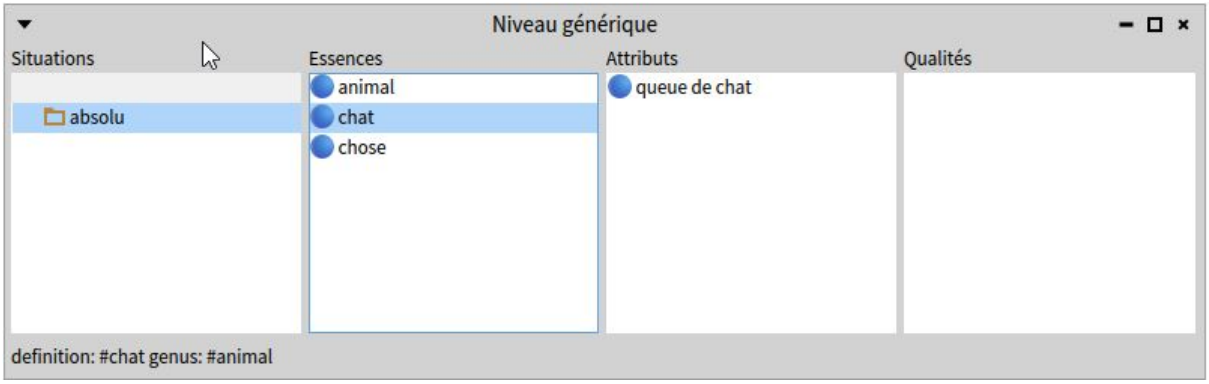
```
1 iceo definition: #animal.
2 iceo definition: #chat genus: animal.
3 iceo definitionAttribut: #queue de: chat.
```

Les essences animal et chat ont été définies dans l'absolu, tandis que l'essence chat constitue la situation de définition de sa queue, ce qui peut être vérifié par l'expression

"`chat getEssenceAttribut: #queue`" :

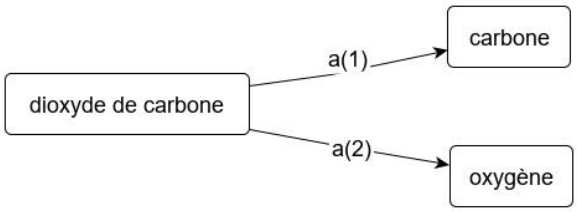


Ceci est confirmé dans un SgBrowser :



Exemple 3 : sur la composition d'une essence à partir d'essences existantes

Cet exemple correspond au graphe suivant :

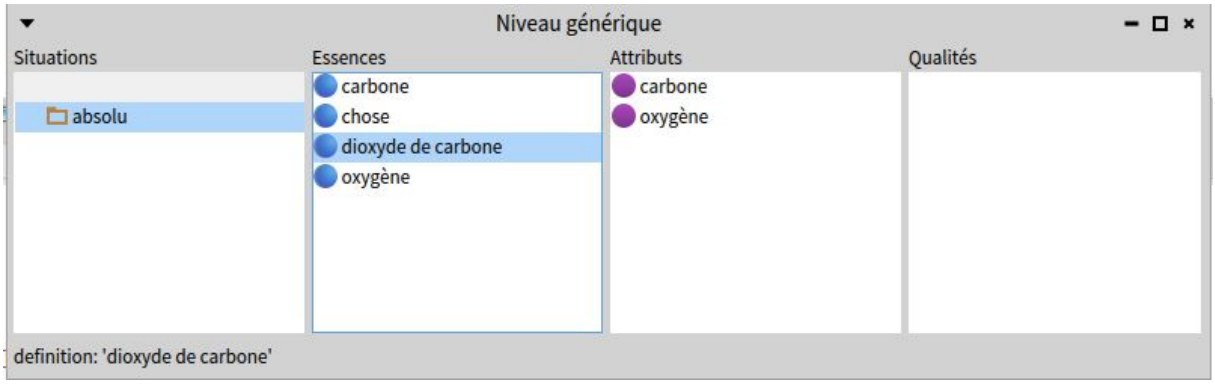


L'essence 'dioxyde de carbone' comporte un atome de carbone et deux atomes d'oxygène qui ne lui sont pas propres :

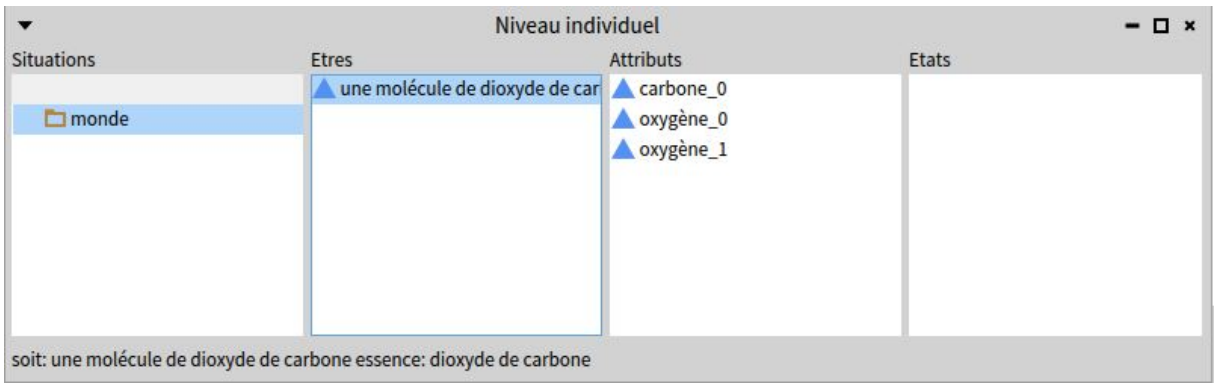
```
1 iceo definition: #oxygène.
2 iceo definition: #carbone.
3 iceo definition: 'dioxyde de carbone'.
4 (absolu get: 'dioxyde de carbone' referenceEssence: oxygène cardinalite: 2.
5 (absolu get: 'dioxyde de carbone' referenceEssence: carbone cardinalite: 1.
6 iceo soit: 'une molécule de dioxyde de carbone' essence: (absolu get: '
    dioxyde de carbone' ).
```

La quatrième ligne indique que l'essence 'dioxyde de carbone' référence l'essence existante oxygène (définie ici dans l'absolu) avec une cardinalité de 2.

Les attributs carbone et oxygène sont affichés avec un petit cercle violet dans un SgBrowser pour indiquer qu'il ne s'agit pas d'attributs propres de dioxyde de carbone :



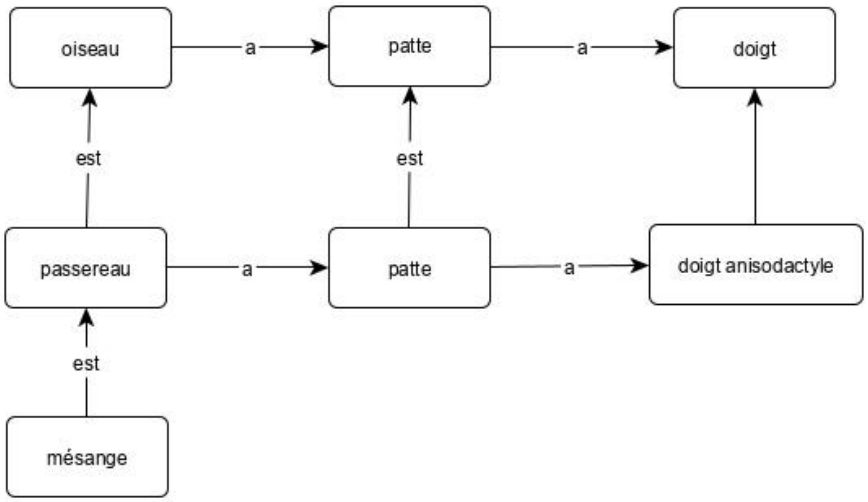
Dans un SiBrowser nous voyons que les êtres attributs de 'une molécule de dioxyde de carbone' ont été créés avec la cardinalité définie au niveau de son essence :



Comme le dioxyde de carbone est un individu, son instanciation a entraîné la création de ses attributs. Le nom des attributs a été généré automatiquement en tenant compte du nom de leur essence.

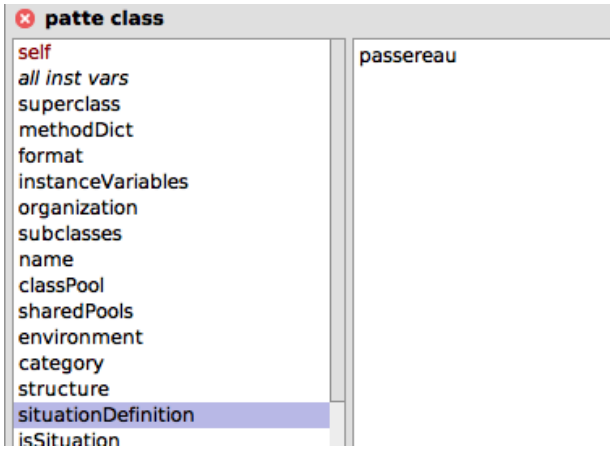
Exemple 4 : sur le principe d'héritage des attributs des essences

Considérons le graphe suivant : :



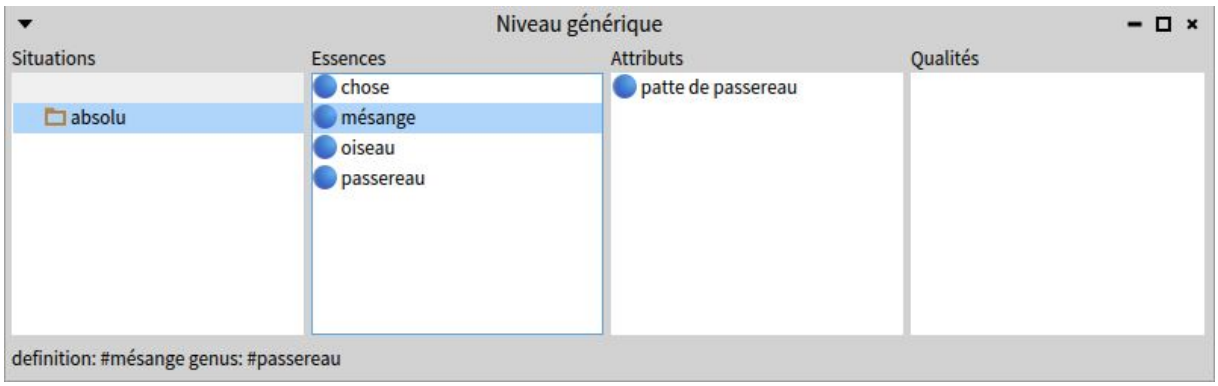

```
1 iceo definition: #oiseau.
2 iceo definition: #passereau genus: oiseau.
3 iceo definition: #mésange genus: passereau.
4 iceo definitionAttribut: #patte de: oiseau.
5 iceo definitionAttribut: #doigt de: (oiseau getEssenceAttribut: #patte).
6 iceo definitionAttribut: #patte de: passereau
7 genus: (oiseau getEssenceAttribut: #patte).
8 iceo definitionAttribut: 'doigt anisodactyle' de: (passereau
    getEssenceAttribut: #patte) genus: ((oiseau getEssenceAttribut: #patte)
    getEssenceAttribut: #doigt) .
```

L'inspection de " `mésange getEssenceAttribut: #patte` " donne :



Il s'agit donc de l'attribut patte de passereau

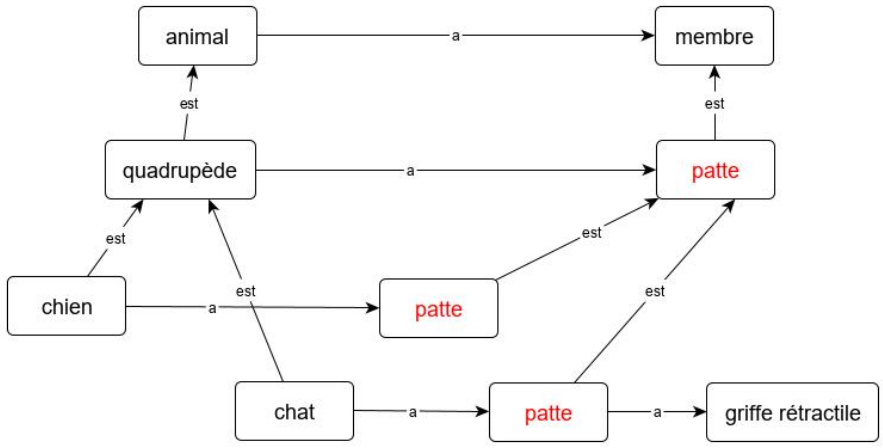
Ceci est confirmé dans un SgBrowser :



L'essence attribut "doigt anisodactyle" de patte de mésange est héritée de patte de passereau, ce qui confirmé par un "Inspect it" de " (mésange getEssenceAttribut: #patte) getEssenceAttribut: 'doigt anisodactyle' "

Exemple 5 : sur le principe d'identification des essences

Considérons le graphe suivant :




```
1 iceo definition: #animal.
2 iceo definition: #quadrupède genus: animal.
3 iceo definition: #chien genus: quadrupède.
4 iceo definition: #chat genus: quadrupède.
5 iceo definitionAttribut: #membre de: animal.
6 iceo definitionAttribut: #patte de: quadrupède genus: (animal
      getEssenceAttribut: #membre).
7 iceo definitionAttribut: #patte de: chien genus: (quadrupède
      getEssenceAttribut: #patte).
8 iceo definitionAttribut: #patte de: chat genus: (quadrupède
      getEssenceAttribut: #patte).
9 iceo definitionAttribut: 'griffe rétractile' de: (chat getEssenceAttribut:
     #patte).
```

On voit dans cet exemple que diverses essences peuvent avoir le même nom dans des situations différentes.

Du fait que différentes essences peuvent avoir le même nom, il est possible de demander la génération d'un identificateur propre à chaque essence, pour vérifier par exemple que l'essence patte de quadrupède est différente de celle de chat et de chien.

Ainsi, les expressions suivantes :

```
'patte de quadrupède : ', (quadrupède getEssenceAttribut: #patte) getId; cr.
'patte de chien : ', (chien getEssenceAttribut: \#patte) getId; cr.
'patte de chat : ', (chat getEssenceAttribut: \#patte) getId; cr
```

donnent :

patte de quadrupède : patte_0

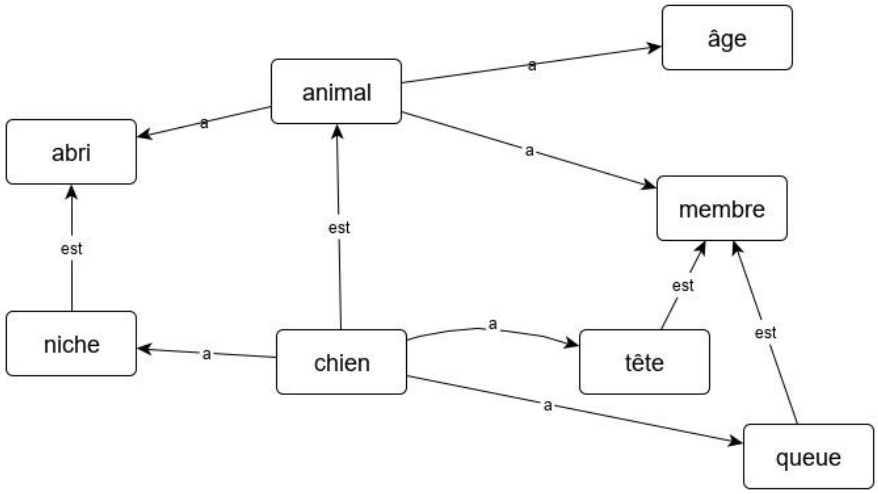
patte de chien : patte_1

patte de chat : patte_2

Exemple 5_2

L'identification d'une essence peut aussi se faire en utilisant le nom de son genus.

Considérons le graphe suivant :




```
1 iceo definition: #animal.
2 iceo definitionAttribut: #âge de: animal.
3 iceo definitionAttribut: #membre de: animal.
4 iceo definitionAttribut: #abri de: animal.
5 iceo definition: #chien genus: animal.
6 iceo definitionAttribut: #niche de: chien genus: (animal getEssenceAttribut
    : #abri).
7 iceo definitionAttribut: #tête de: chien genus: (animal getEssenceAttribut:
     #membre).
8 iceo definitionAttribut: #queue de: chien genus: (animal getEssenceAttribut
    : #membre).
```

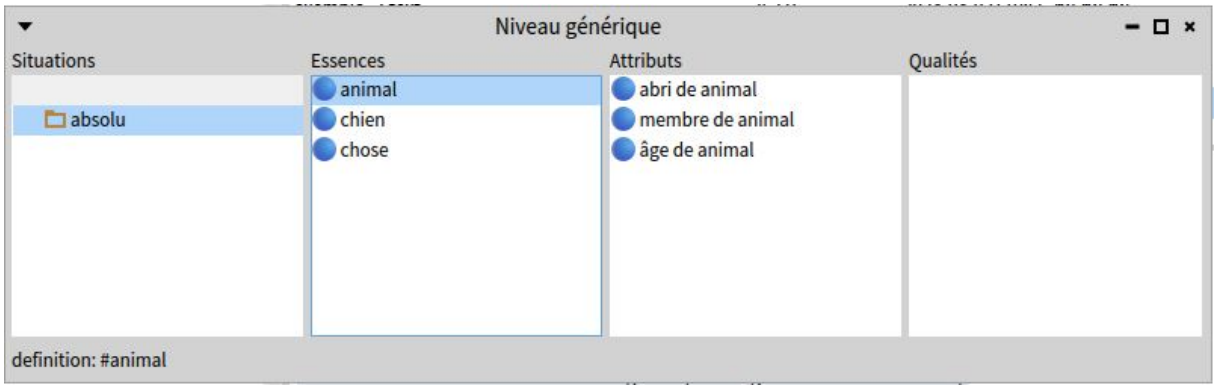
L'expression " `chien getEssenceAttribut: #âge` " donne : âge

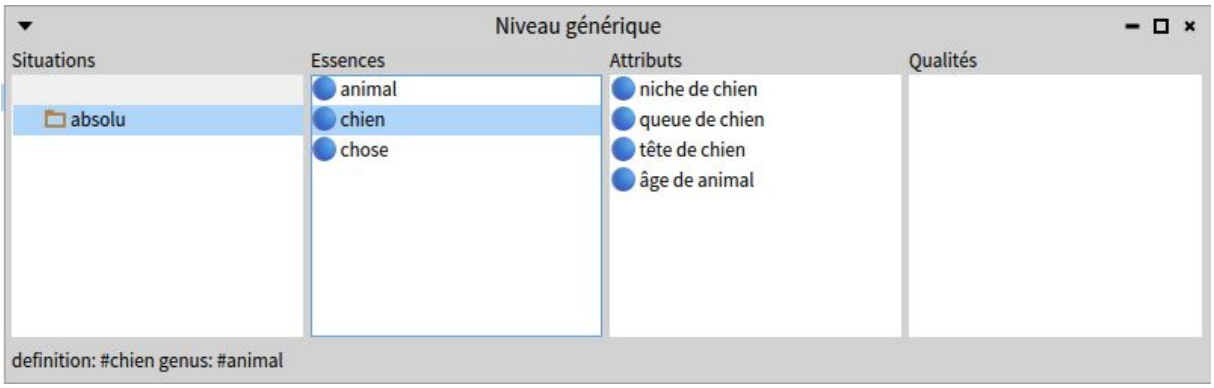
Il s'agit de l'attribut âge hérité de l'essence animal.

L'expression " `chien getEssenceAttribut: #abri` " donne: niche

et l'expression " `chien getEssencesAttributs: #membre` " donne : an OrderedCollection(queue tête)

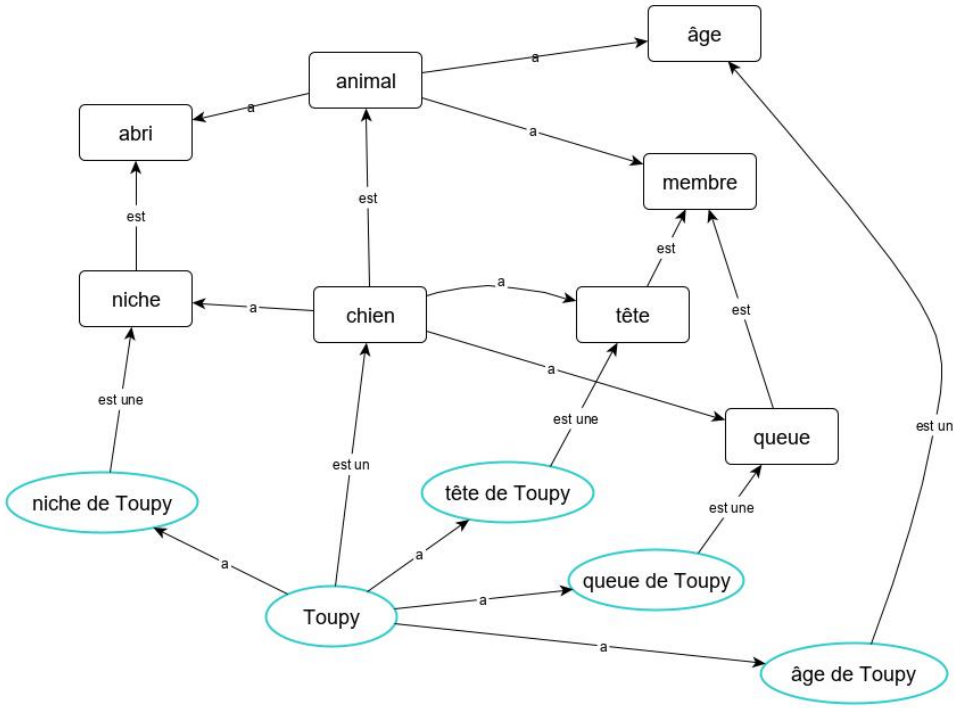
Ce qui se vérifie dans les fenêtres suivantes :


Niveau générique
Situations
Essences
Attributs
Qualités
absolu
animal
chien
chose
abri de animal
membre de animal
âge de animal
definition: #animal


Niveau générique
Situations
Essences
Attributs
Qualités
absolu
animal
chien
chose
niche de chien
queue de chien
tête de chien
âge de animal
definition: #chien genus: #animal

Exemple 6 : sur le principe d'identification des êtres

Considérons le graphe suivant :

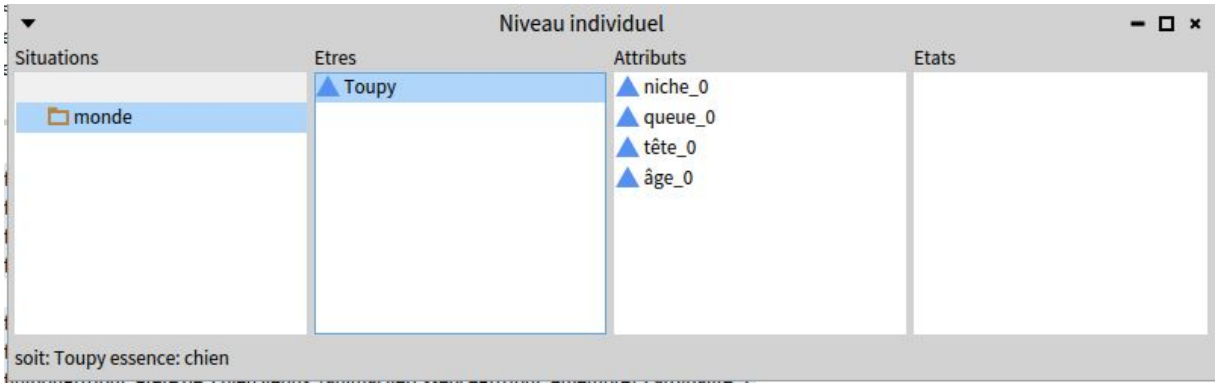



```
1 iceo definition: #animal.
2 iceo definitionAttribut: #âge de: animal cardinalite: 1.
3 iceo definitionAttribut: #membre de: animal.
4 iceo definitionAttribut: #abri de: animal.
5 iceo definition: #chien genus: animal.
6 iceo definitionAttribut: #niche de: chien genus: (animal getEssenceAttribut
    : #abri) cardinalite: 1.
7 iceo definitionAttribut: #tête de: chien genus: (animal getEssenceAttribut:
     #membre) cardinalite: 1.
8 iceo definitionAttribut: #queue de: chien genus: (animal getEssenceAttribut
    : #membre) cardinalite: 1.
9 iceo soit: #Toupy essence: chien.
```

L'expression " `(monde get: #Toupy) getEtresAttributs` " donne : an OrderedCollection(a niche a queue a tête a âge)

Les attributs de Toupy ont été créés automatiquement car il s'agit par défaut d'un individu.

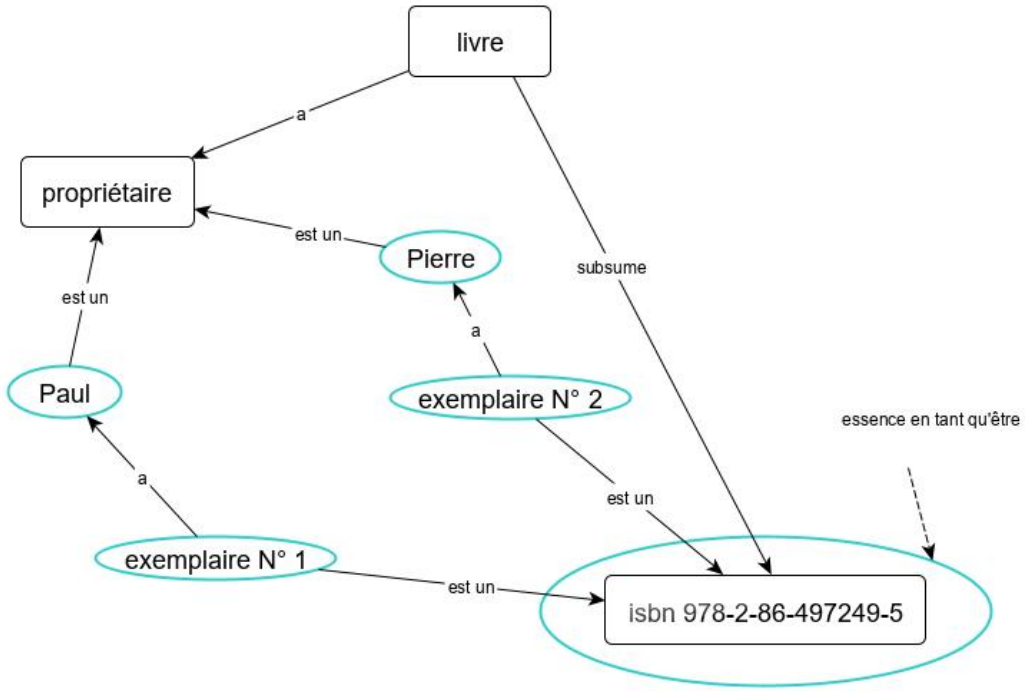
Leur nom a été créé par un générateur de symboles en se basant sur le nom de leur essence.



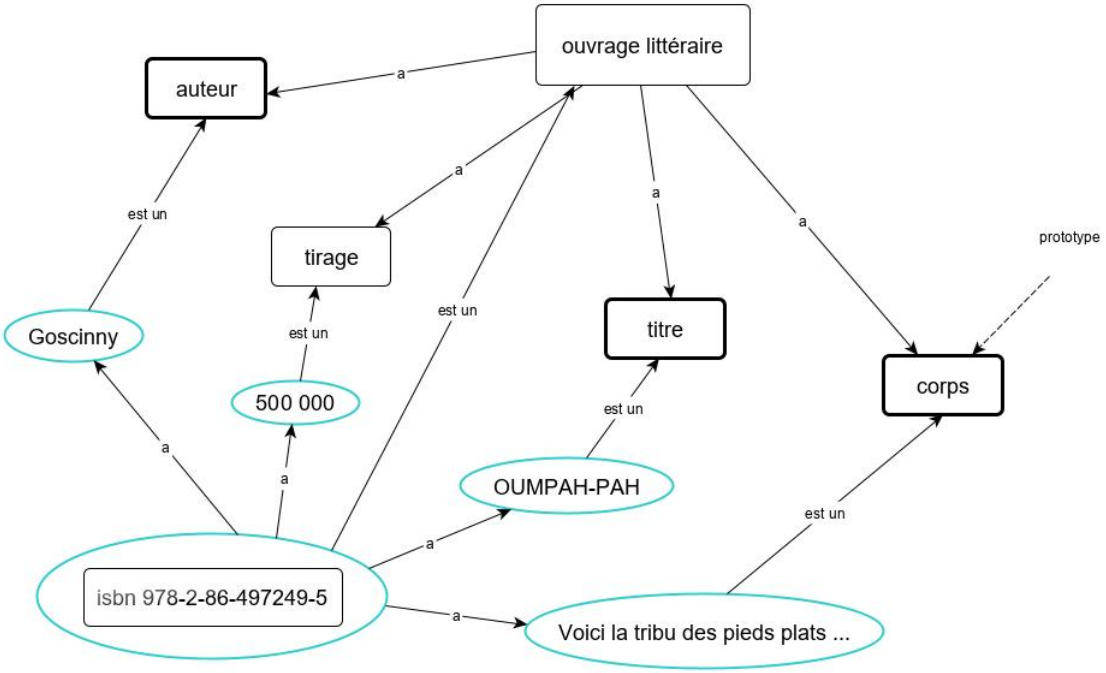
L'expression " `((monde get: #Toupy) getEtreAttribut: #abri) getNom` " donne : #niche_0

Exemple 7 : sur la notion de méta essence (essence d'une essence)

Le graphe suivant correspond à la définition de l'essence 'isbn 978-2-86-497249-5' subsumée par l'essence livre, avec deux exemplaires dont les propriétaires respectifs sont Paul et Pierre :



En tant qu'être, l'essence "isbn 978-2-86-497249-5" est instance de l'essence "ouvrage littéraire" (sa méta essence" :

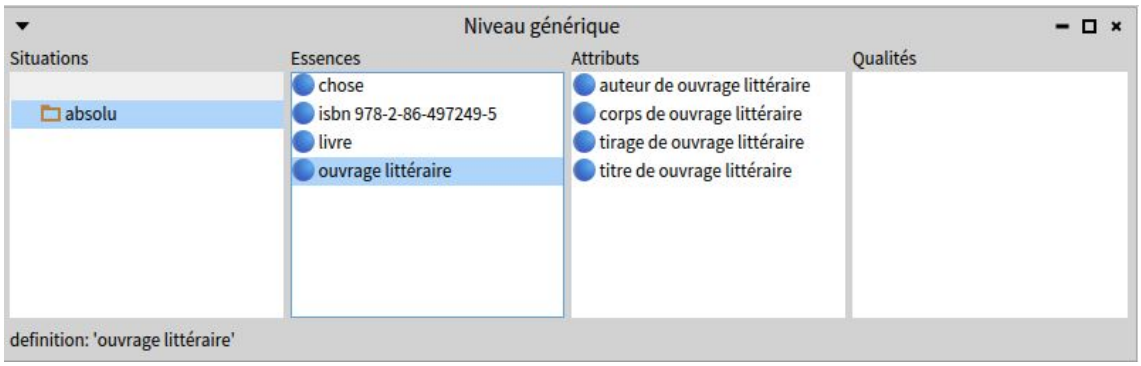


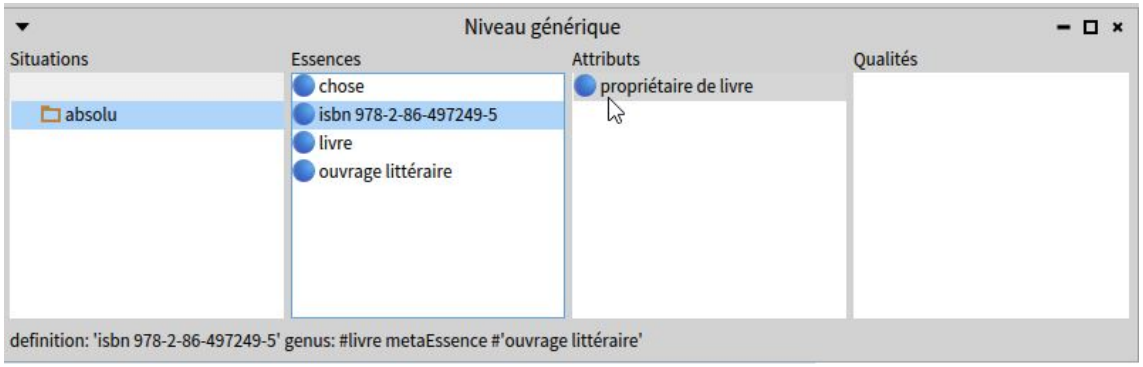
Voici la représentation du contenu de ces deux graphes dans ICEO :

```
1  iceo definition: 'ouvrage littéraire'.
2  iceo definitionAttribut: #corps de: (absolu get: 'ouvrage littéraire')
       isPrototype: true.
3  iceo definitionAttribut: #tirage de: (absolu get: 'ouvrage littéraire').
4  iceo definitionAttribut: #auteur de: (absolu get: 'ouvrage littéraire')
       isPrototype: true.
5  iceo definitionAttribut: #titre de: (absolu get: 'ouvrage littéraire')
       isPrototype: true.
6  iceo definition: #livre.
7  iceo definitionAttribut: #propriétaire de: livre.
8  iceo definition: 'isbn 978-2-86-497249-5' genus: livre metaEssence: (absolu
       get: 'ouvrage littéraire').
9  (absolu get: 'isbn 978-2-86-497249-5') attributionEtre: '500000' .
10 essence: ((absolu get: 'ouvrage littéraire') getEssenceAttribut: #tirage).
11 (absolu get: 'isbn 978-2-86-497249-5') attributionEtre: #Goscinny.
12 essence: ((absolu get: 'ouvrage littéraire') getEssenceAttribut: #auteur).
13 (absolu get: 'isbn 978-2-86-497249-5') attributionEtre: 'OUMPAH-PAH'.
14 essence: ((absolu get: 'ouvrage littéraire') getEssenceAttribut: #titre).
15 (absolu get: 'isbn 978-2-86-497249-5') attributionEtre: 'Voici la tribu des
       pieds plats...'.
16 essence: ((absolu get: 'ouvrage littéraire') getEssenceAttribut: #corps).
17 iceo soit: #Paul essence: (livre getEssenceAttribut: #propriétaire).
18 iceo soit: #Exemplaire\_1 essence: (absolu get: 'isbn 978-2-86-497249-5') .
19 (monde get: #Exemplaire\_1) attributionEtre: (monde get: #Paul).
20 iceo soit: #Pierre essence: (livre getEssenceAttribut: #propriétaire).
21 iceo soit: #Exemplaire\_2 essence: (absolu get: 'isbn 978-2-86-497249-5').
22 (monde get: #Exemplaire\_2) attributionEtre: (monde get: #Pierre).
```

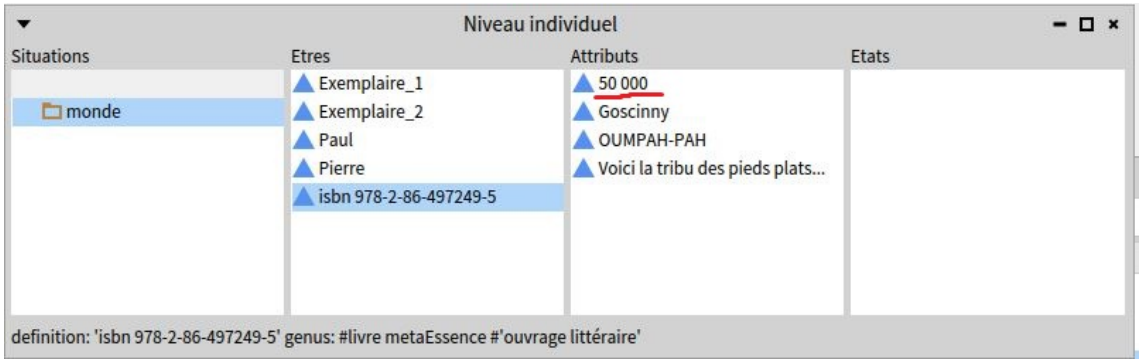
La ligne 8 définit l'essence "isbn 978-2-86-497249-5" comme subsumée par livre et instance d'ouvrage littéraire (sa méta essence).

Dans un SgBrowser nous pouvons visualiser les attributs des essences ouvrage littéraire et isbn 978-2-86-497249-5 :

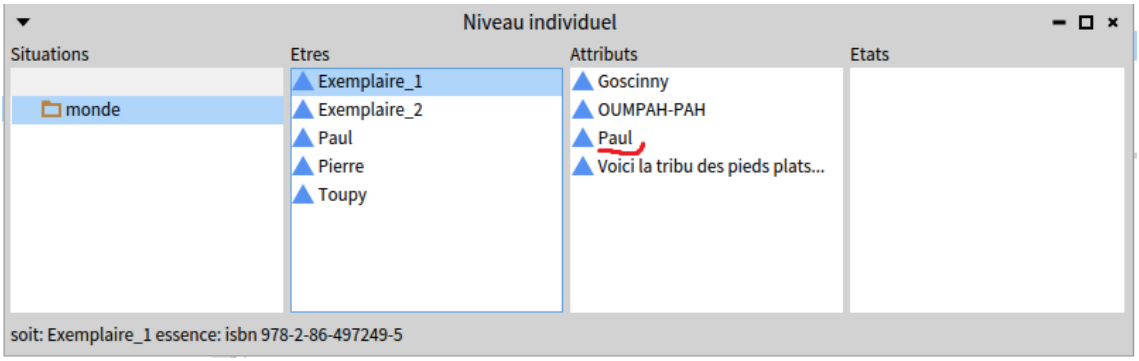


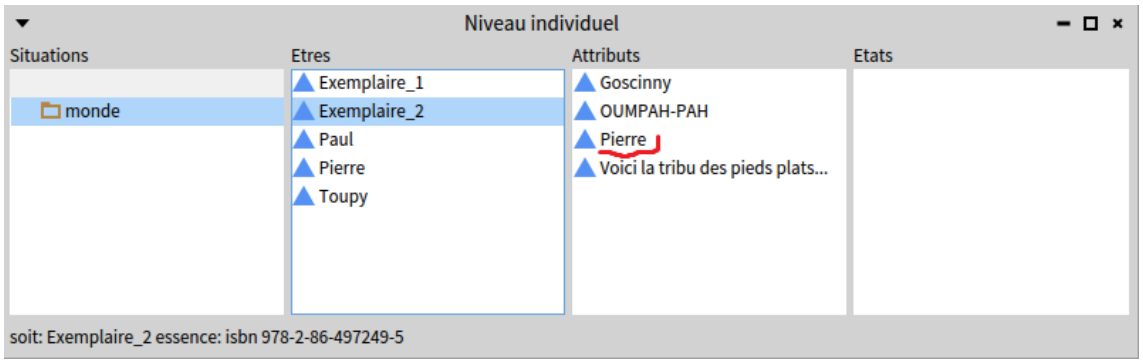


et dans un SiBrowser les attributs de l'essence isbn 978-2-86-497249-5 en tant qu'être :



et les attributs des exemplaire 1 et 2 :





On voit que les deux exemplaires ont les mêmes attributs titre, auteur et corps (qualifiés de prototypes au niveau de ouvrage littéraire) mais que leur attribut propriétaire, que ne possède pas isbn 978-2-86-497249-5, est différent.

Par contre, il n'ont pas l'attribut tirage que possède isbn 978-2-86-497249-5.

Notons q'une petite anomalie apparait dans cet exemple, car il n'est pas vraiment correct de considérer Pierre ou Paul comme attributs d'un exemplaire du livre.

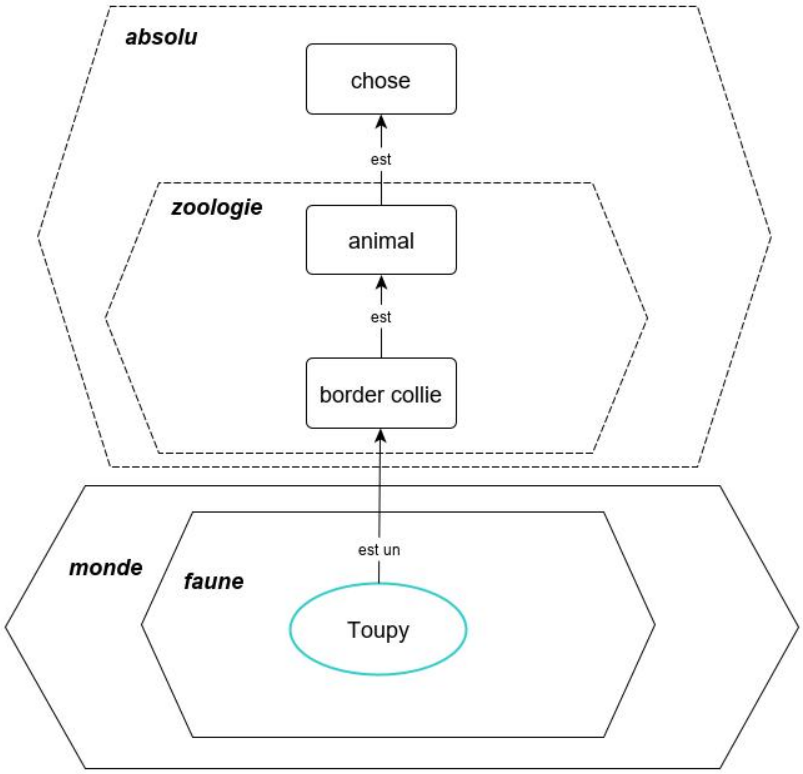
En fait, ce sont Pierre et Paul qui sont chacun propriétaire d'un exemplaire du livre.

Dire qu'un livre a un propriétaire est en fait un raccourci de langage pour exprimer qu'un exemplaire du livre appartient à une personne qui en est propriétaire.

Nous verrons comment corriger ce point avec l'exemple 14.

Exemple 8 : sur les notions de situation générique et de situation individuelle

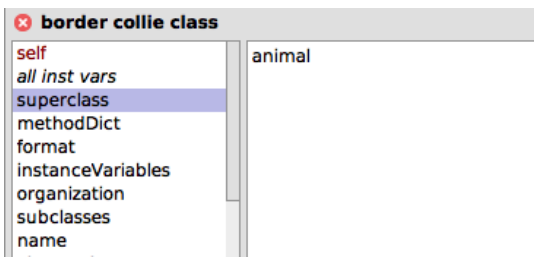
Considérons le graphe suivant :



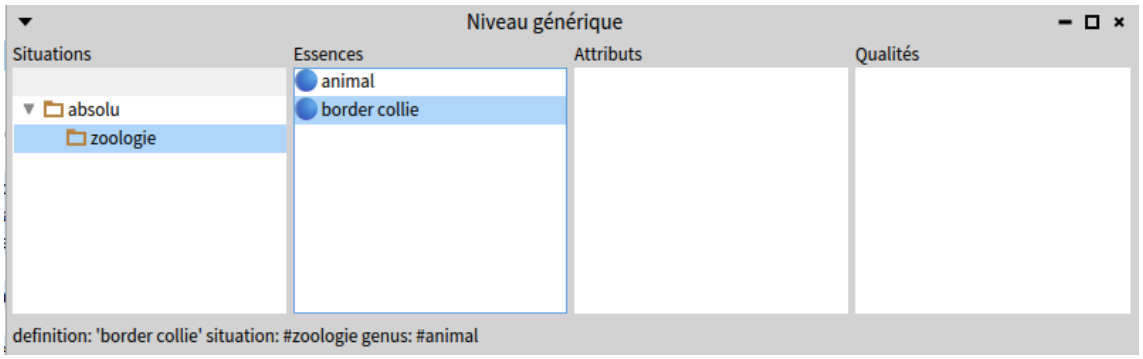

```
1 iceo definitionSituation: #zoologie.
2 iceo definition: #animal situation: zoologie.
3 iceo definition: 'border collie' situation: zoologie genus: (zoologie get:
    #animal).
4 iceo soit: #faune situationGenerique: zoologie.
5 iceo soit: #Toupy essence: (zoologie get: 'border collie')
    situationIndividuelle: (monde getSituation: #faune).
```

L'essence nommée "border collie" est subsumée par l'essence animal dans une situation générique nommée "zoologie", et un être "Toupy" est créé dans une situation individuelle nommée "faune". La situation générique zoologie est incluse dans l'absolu et la situation individuelle faune, instance de la situation zoologie, est incluse dans le monde.

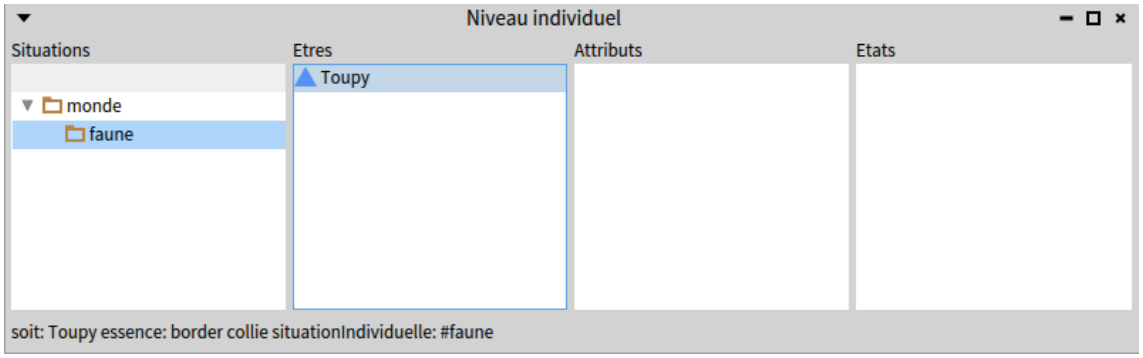
L'inspection de "border collie" en utilisant l'expression "zoologie get: 'border collie' " permet de vérifier qu'elle est bien subsumée par l'essence animal :



et sa situation générique est zoologie :

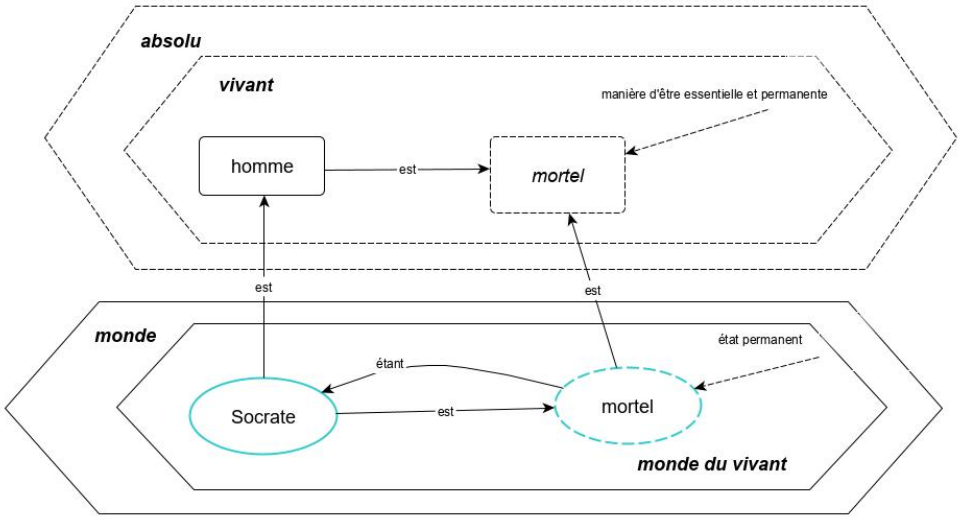


L'expression " `(monde getSituation: #faune) get: #Toupy` " permet de vérifier que Toupy est bien une instance de border collie présente dans la situation individuelle faune, ce qui se vérifie dans la fenêtre suivante :



Exemple 9 : sur la notion de manière d'être essentielle et permanente

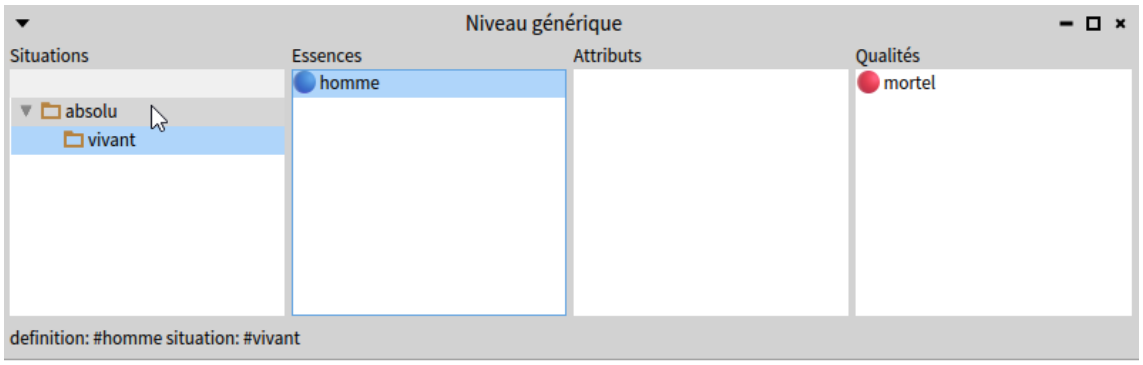
Considérons le graphe suivant :




```
1 iceo definitionSituation: #vivant.
2 iceo definition: #homme situation: vivant.
3 iceo definitionQualiteEssentielle: #mortel pour: (vivant get: #homme)
      effectivite: #permanente.
4 iceo soit: 'monde du vivant' situationGenerique: vivant.
5 iceo soit: #Socrate essence: (vivant get: #homme) situationIndividuelle: (
      monde get: 'monde du vivant').
```

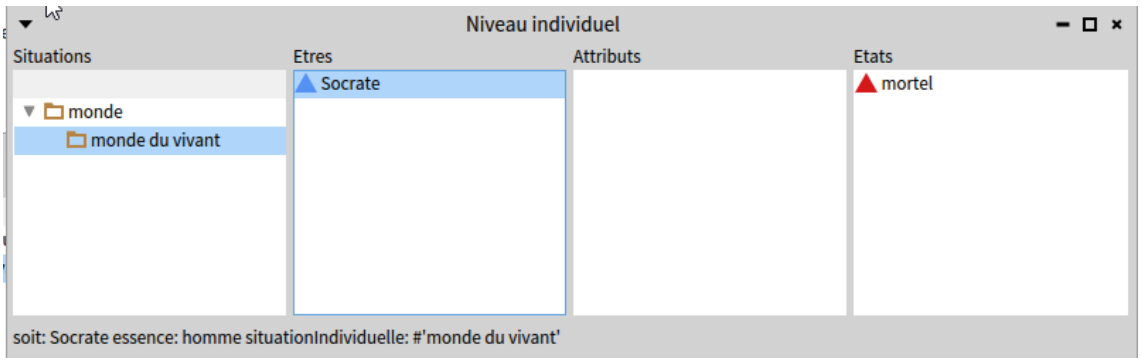
La troisième ligne définit la manière d'être "mortel" comme essentielle et permanente pour homme.

C'est ce qui se vérifie dans la fenêtre suivante :



La petite icône rouge associée à la qualité "mort" indique qu'il s'agit d'une qualité essentielle.

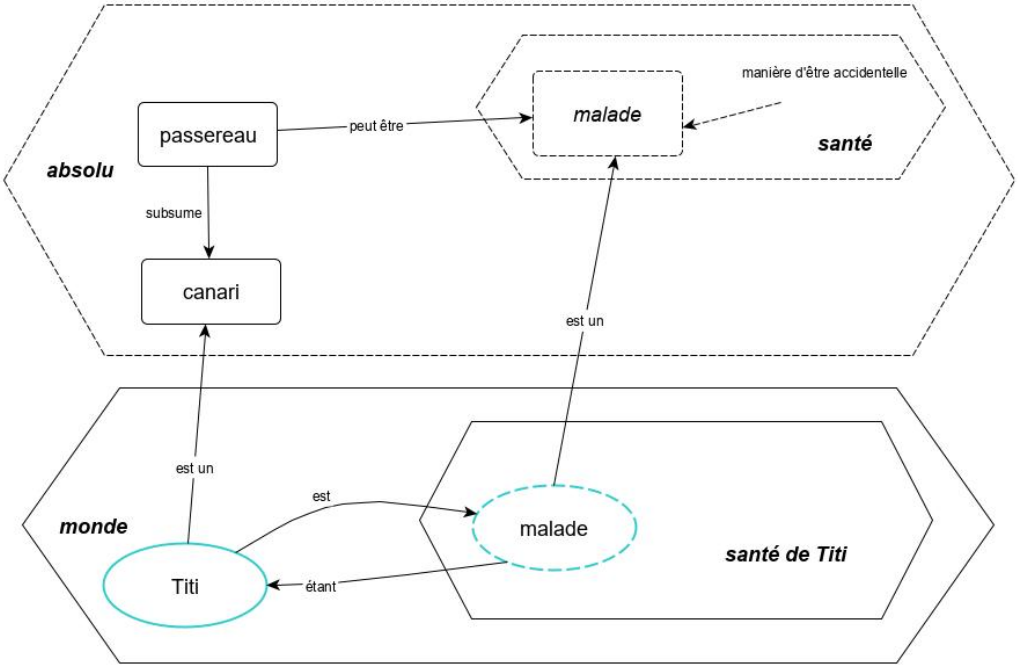
La fenêtre suivante montre que Socrate, en tant qu'homme, est bien mortel :



Ainsi, qu'Aristote se rassure, Socrate en tant qu'homme était bien mortel !

Exemple 10 : sur la notion de manière d'être accidentelle

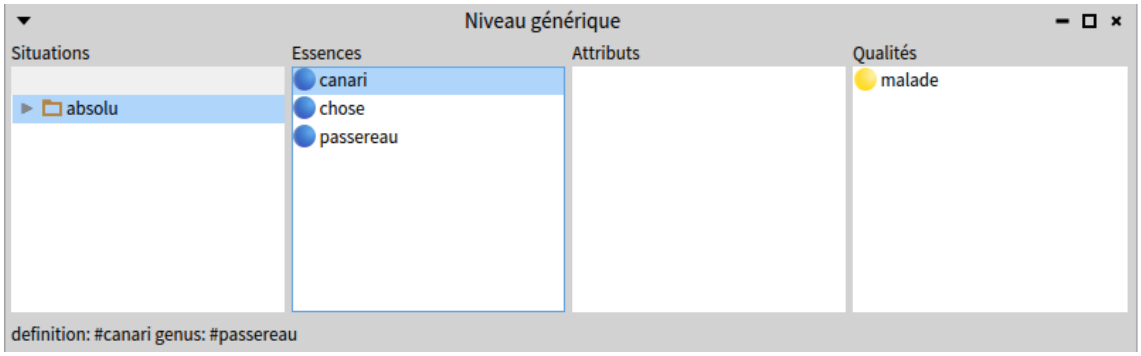
Considérons le graphe suivant :




```
1  iceo definition: #passereau.
2  iceo definition: #canari genus: passereau.
3  iceo definitionSituation: #santé.
4  iceo definitionQualite: #malade situation: santé. passereau peutEtre:
           (santé get: #malade).
5  iceo soit: #Titi essence: canari.
6  iceo soit: 'santé de Titi' situationGenerique: santé.
7  (monde get: #Titi) affecteEtat: (santé get: #malade) dansSituation: (monde
       get: 'santé de Titi').
```

L'essence passereau peut avoir la manière d'être accidentelle "malade" (statut par défaut de la manière d'être "malade" définie à la quatrième ligne).

Cette manière d'être est héritée par l'essence canari qui est subsumée par passereau.



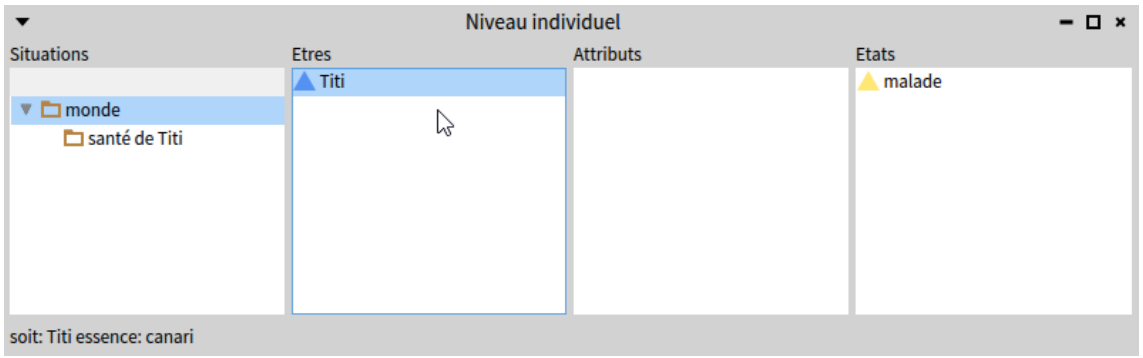
La petite icône jaune associée à la qualité "malade" indique qu'il s'agit d'une qualité accidentelle.

Pour que Titi soit malade, il faut que cet état lui soit explicitement affecté, car il s'agit d'une manière d'être accidentelle.

C'est ce qui est fait dans la dernière ligne.

L'état malade de Titi est instance de la manière d'être malade.

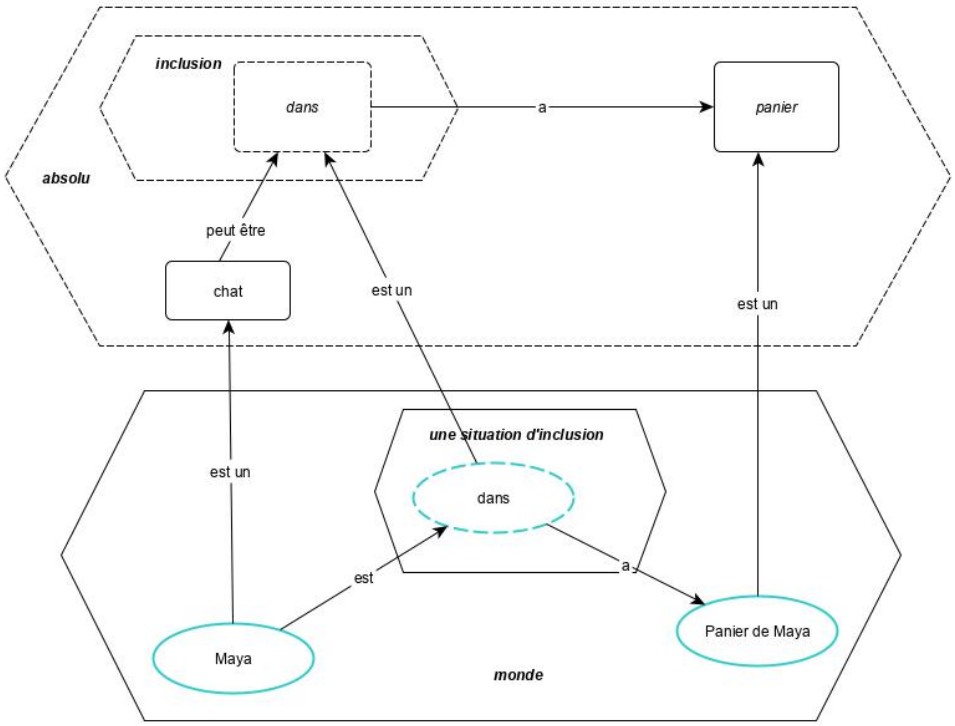
C'est ce que confirme la fenêtre suivante :



Souhaitons à Titi un prompt rétablissement.

Exemple 11 : sur les attributs d'une manière d'être

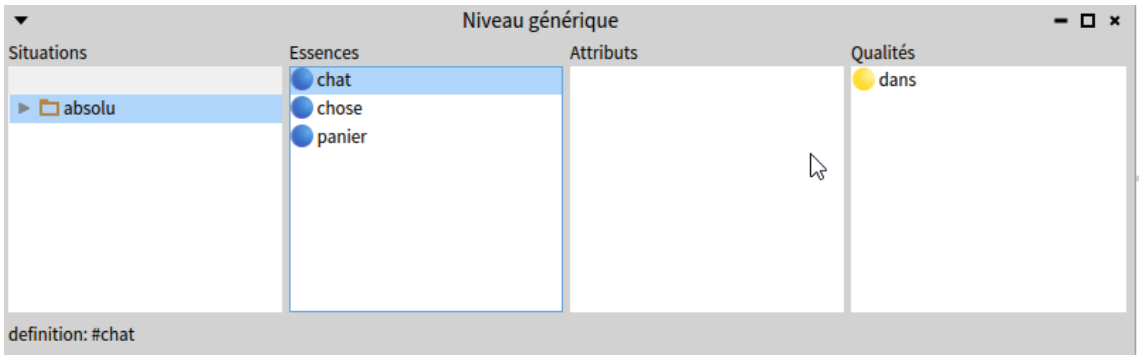
Considérons le graphe suivant:



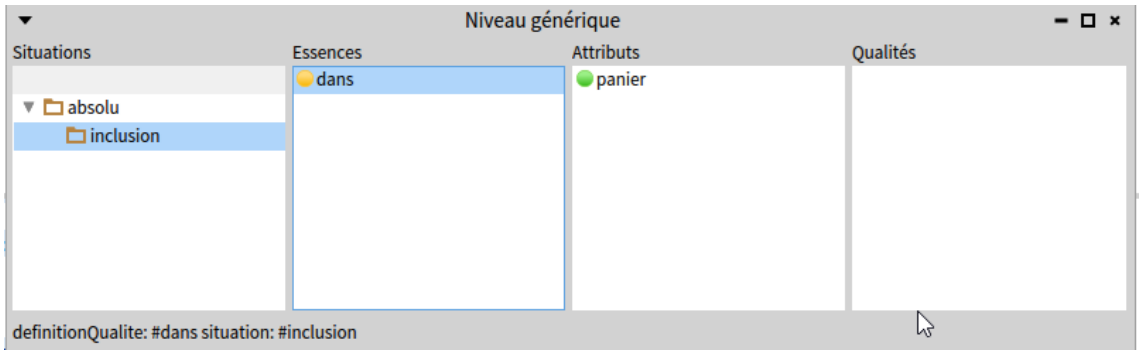
Il est la représentation de la phrase : "La chatte Maya est dans son panier"

```
1  iceo definition: #chat.
2  iceo definition: #panier.
3  iceo definitionSituation: #inclusion.
4  iceo definitionQualite: #dans situation: inclusion.
5  (inclusion get: #dans) referenceEssence: panier.
6  chat peutEtre: (inclusion get: #dans).
7  iceo soit: 'une situation d''inclusion' situationGenerique: inclusion.
8  iceo soit: #Maya essence: chat.
9  iceo soit: 'Panier de Maya' essence: panier.
10 (monde get: #Maya) affecteEtat: (inclusion get: #dans) dansSituation: (
       monde get: 'une situation d''inclusion).
11 ((monde get: #Maya) getEtat: #dans) attributionEtre: (monde get: 'Panier de
       Maya' ).
```

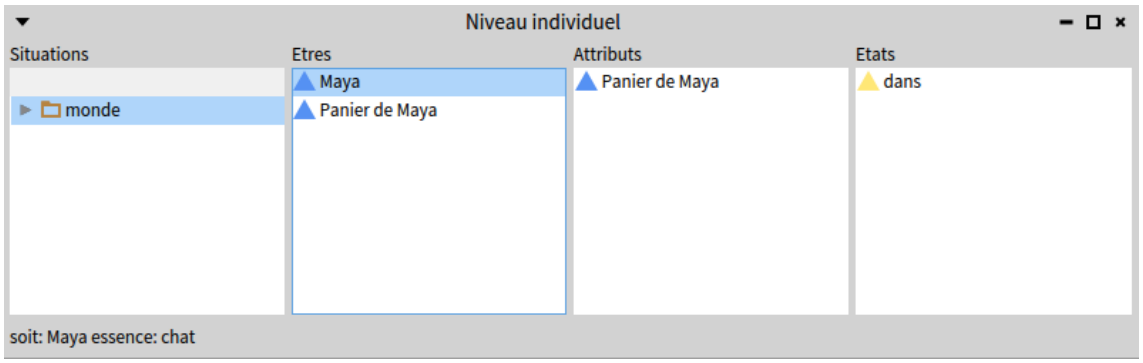
Les fenêtres suivantes montrent que "dans" est une qualité accidentelle de chat :

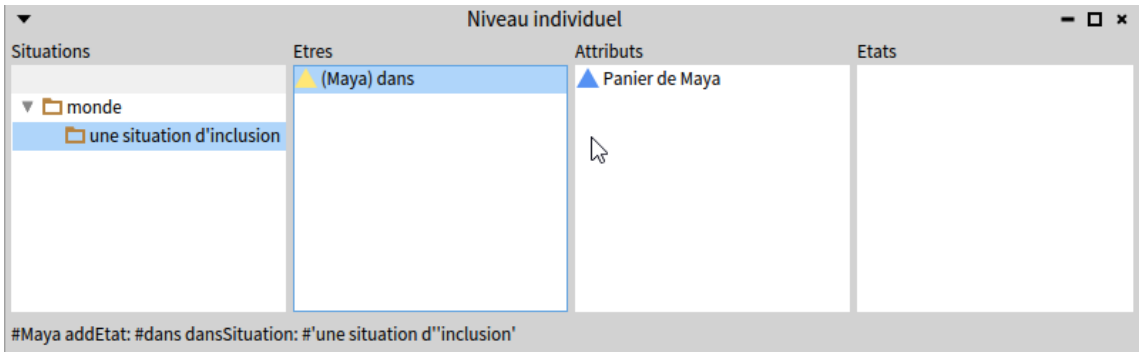


et que l'essence panier est attribut de "dans" :



et celles-ci montrent que Maya est "dans" son panier :





L'état "dans" qui s'affiche en tant qu'être (ce qui est normal, car un état est un être !) est précédé du nom de son étant (l'être qui est dans cet état) placé entre parenthèses.

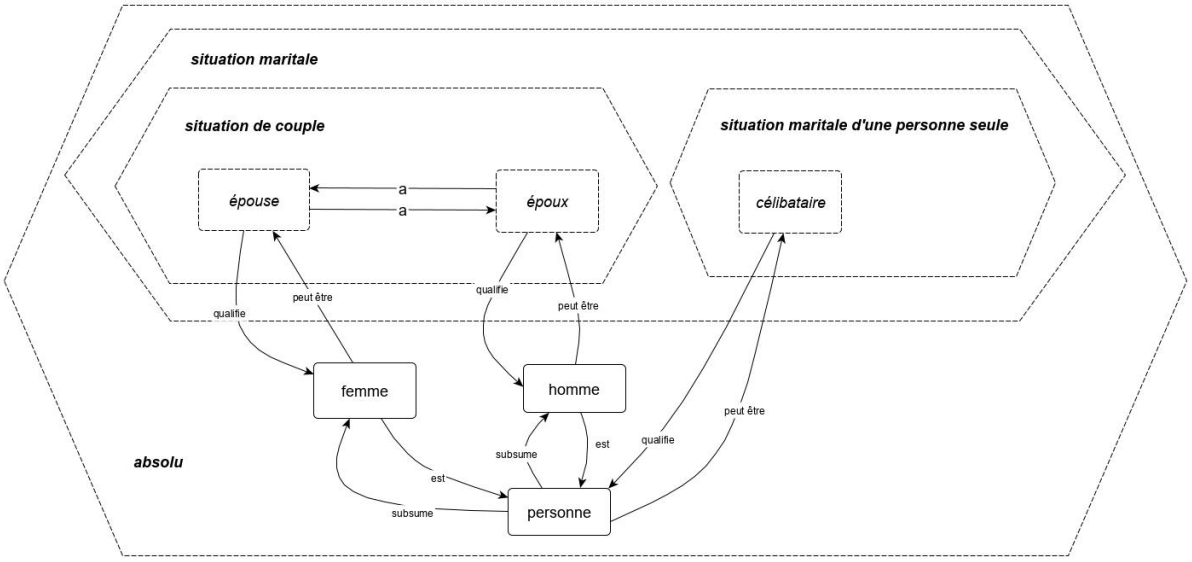
L'expression " `(monde get: #Maya) getEtresAttributsEnTantQue: (inclusion get: #dans)` "

donne : an OrderedCollection(a panier)

Suivant la même logique, nous pourrions représenter toutes les prépositions de la langue française (sur, avec, entre, ...)

Exemple 12 : sur les relations entre manières d'être

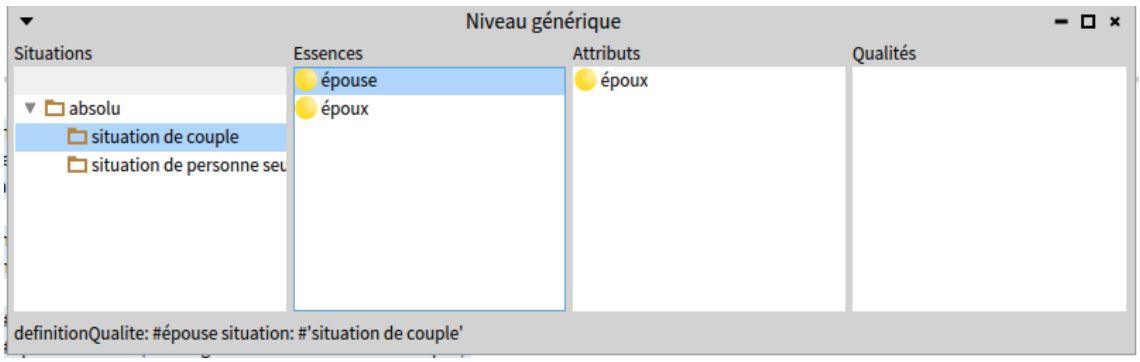
Considérons le graphe suivant :

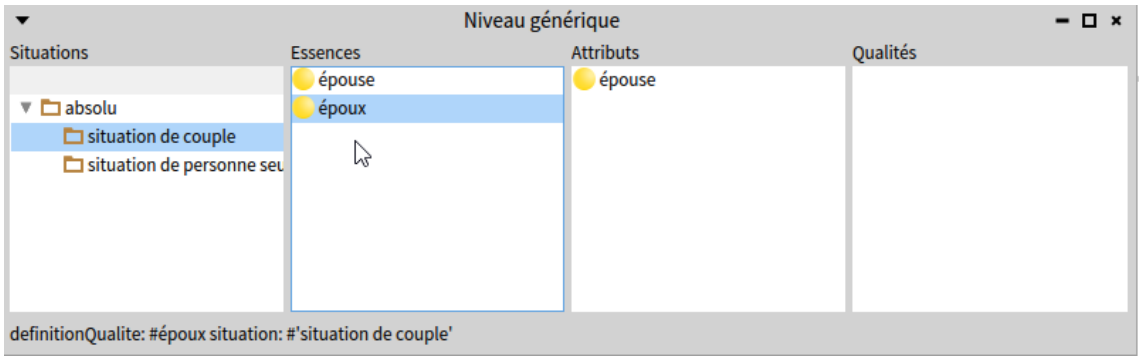



```
1  iceo definition: #personne
2  iceo definition: #femme genus: personne
3  iceo definition: #homme genus: personne
4  iceo definition: #maire genus: personne
5  iceo definitionSituation: 'situation de couple'
6  iceo definitionSituation: 'situation de personne seule'
7  iceo definitionQualite: #épouse situation: (absolu getSituation: 'situation
       de couple')
8  iceo definitionQualite: #époux situation: (absolu getSituation: 'situation
      de couple')
9  iceo definitionQualite: #célibataire situation: (absolu getSituation: '
      situation de personne seule')
10 ((absolu getSituation: 'situation de couple') get: #épouse)
      associationQualite: ((absolu getSituation: 'situation de couple')
              get: #époux).
11 femme peutEtre: ((absolu getSituation: 'situation de couple') get: #épouse)
      .
12 femme peutEtre: ((absolu getSituation: 'situation de personne seule') get:
      #célibataire).
13 homme peutEtre: ((absolu getSituation: 'situation de couple') get: #époux).
14 homme peutEtre: ((absolu getSituation: 'situation de personne seule') get:
      #célibataire).
```

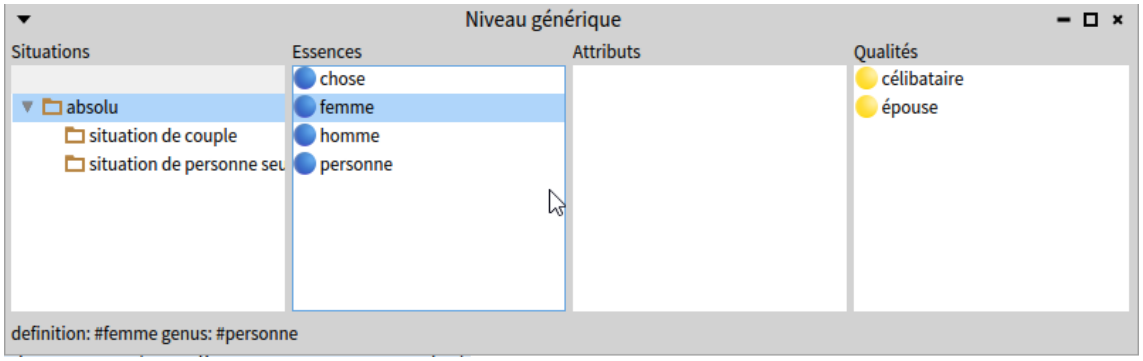
La ligne 10 définit une relation bidirectionnelle entre les manières d'être épouse et époux.

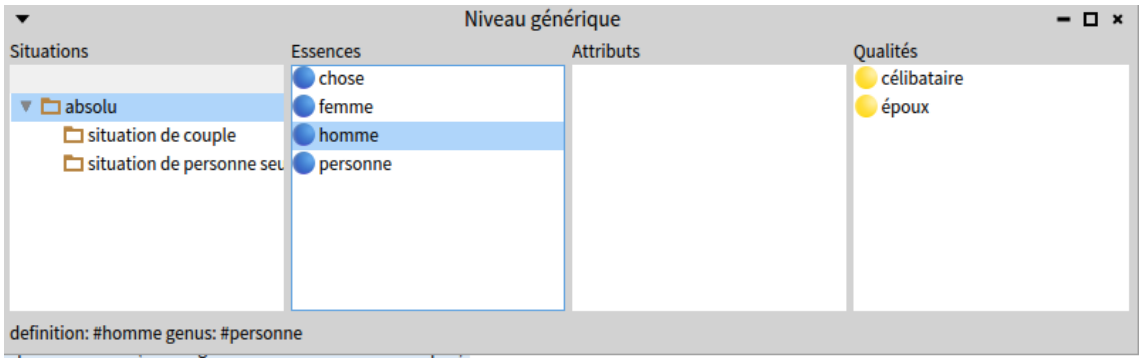
Ceci peut être vérifié dans les fenêtres suivantes :





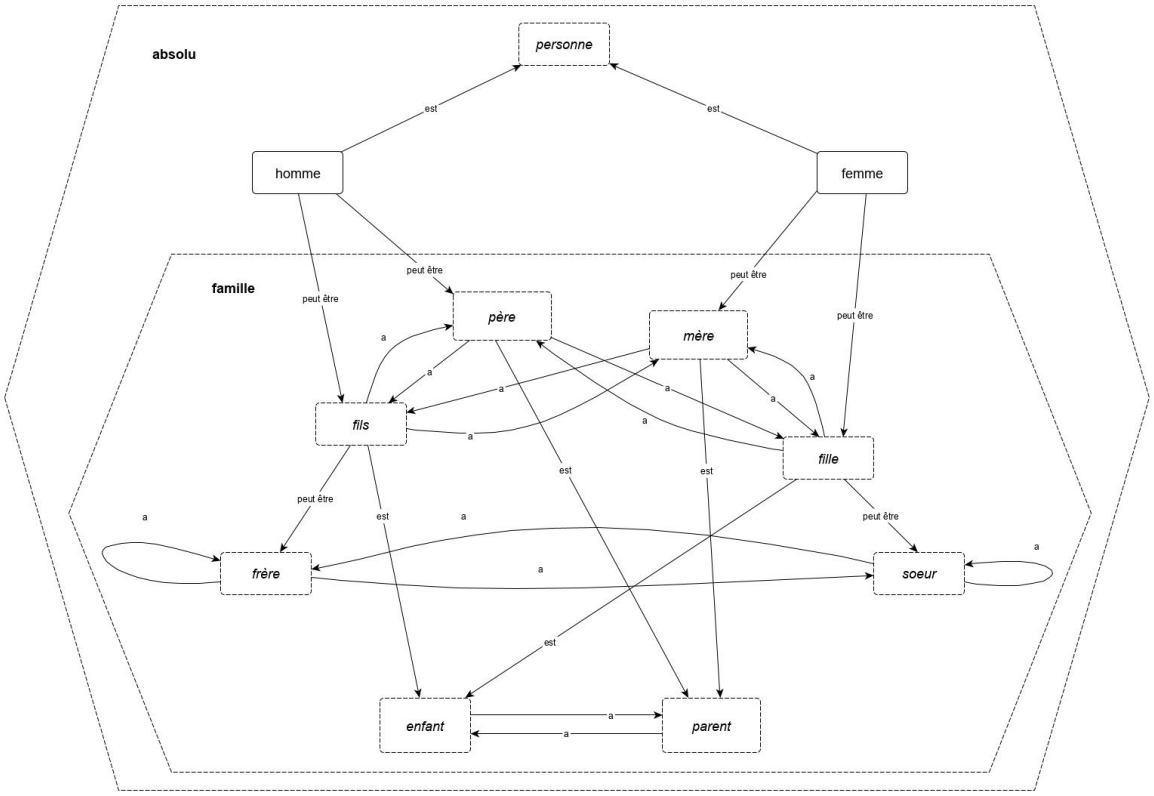
Les manières d'être possibles pour homme et femme apparaissent dans les fenêtres suivantes :





Exemple 13 : sur la subsomption des manières d'être

Considérons le graphe suivant :



La représentation dans ICEO est la suivante :

```
1  iceo definition: #personne.
2  iceo definition: #femme genus: personne.
3  iceo definition: #homme genus: personne.
4  iceo definitionSituation: 'famille'.
5  iceo definitionQualite: #parent situation: (absolu getSituation: #famille).
6  iceo definitionQualite: #enfant situation: (absolu getSituation: #famille).
7  iceo definitionQualite: #soeur situation: (absolu getSituation: #famille).
8  iceo definitionQualite: #frère situation: (absolu getSituation: #famille).
9  iceo definitionQualite: #fils situation: (absolu getSituation: #famille)
       genus: ((absolu getSituation: #famille) get: #enfant).
10 iceo definitionQualite: #père situation: (absolu getSituation: #famille)
       genus: ((absolu getSituation: #famille) get: #parent).
11 iceo definitionQualite: #fille situation: (absolu getSituation: #famille)
       genus: ((absolu getSituation: #famille) get: #enfant).
12 iceo definitionQualite: #mère situation: (absolu getSituation: #famille)
       genus: ((absolu getSituation: #famille) get: #parent).
13 ((absolu getSituation: #famille) get: #parent) associationQualite: ((absolu
       getSituation: #famille) get: #enfant).
14 ((absolu getSituation: #famille) get: #père) associationQualite: ((absolu
       getSituation: #famille) get: #fils).
15 ((absolu getSituation: #famille) get: #père) associationQualite: ((absolu
       getSituation: #famille) get: #fille).
16 ((absolu getSituation: #famille) get: #mère) associationQualite: ((absolu
       getSituation: #famille) get: #fils).
17 ((absolu getSituation: #famille) get: #mère) associationQualite: ((absolu
       getSituation: #famille) get: #fille).
18 ((absolu getSituation: #famille) get: #frère) associationQualite: ((absolu
```

```
    getSituation: #famille) get: #soeur).
19 ((absolu getSituation: #famille) get: #frère) associationQualite: ((absolu
    getSituation: #famille) get: #frère).
20 ((absolu getSituation: #famille) get: #soeur) associationQualite: ((absolu
    getSituation: #famille) get: #soeur).
21 homme peutEtre: ((absolu getSituation: #famille) get: #père).
22 homme peutEtre: ((absolu getSituation: #famille) get: #fils).
23 femme peutEtre: ((absolu getSituation: #famille) get: #mère).
24 femme peutEtre: ((absolu getSituation: #famille) get: #fille).
25 ((absolu getSituation: #famille) get: #fille) peutEtre: ((absolu
    getSituation: #famille) get: #soeur).
26 ((absolu getSituation: #famille) get: #fils) peutEtre: ((absolu
    getSituation: #famille) get: #frère).
```

Dans cet exemple, certaines manières d'être en subsument d'autres.

Ainsi " `((absolu getSituation: #famille) get: #fils) getGenus` " donne : enfant

Notons également qu'une manière d'être peut adopter d'autres manières d'être.

Ainsi " `((absolu getSituation: #famille) get: #fils) getQualites` " donne : an OrderedCollection(frère)

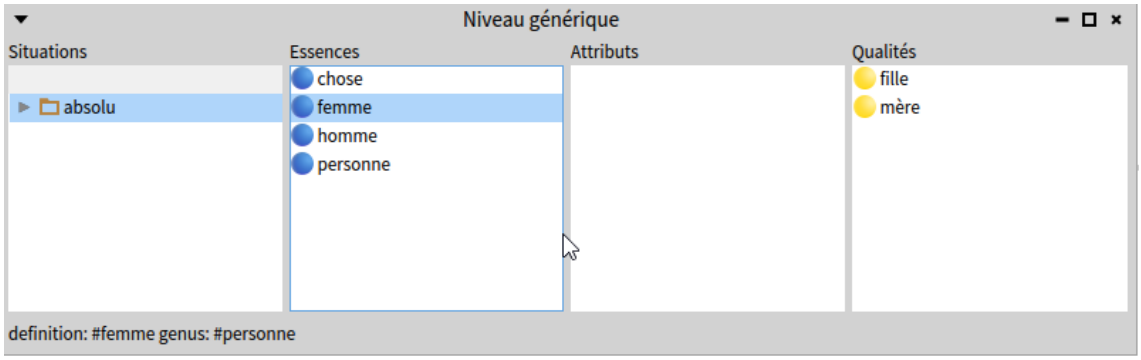
Et, comme nous l'avons déjà vu, une manière d'être peut avoir pour attributs d'autres manières d'être.

Ainsi, " `((absolu getSituation: #famille) get: #frère) getDifferentia` " donne : an OrderedCollection(soeur frère)

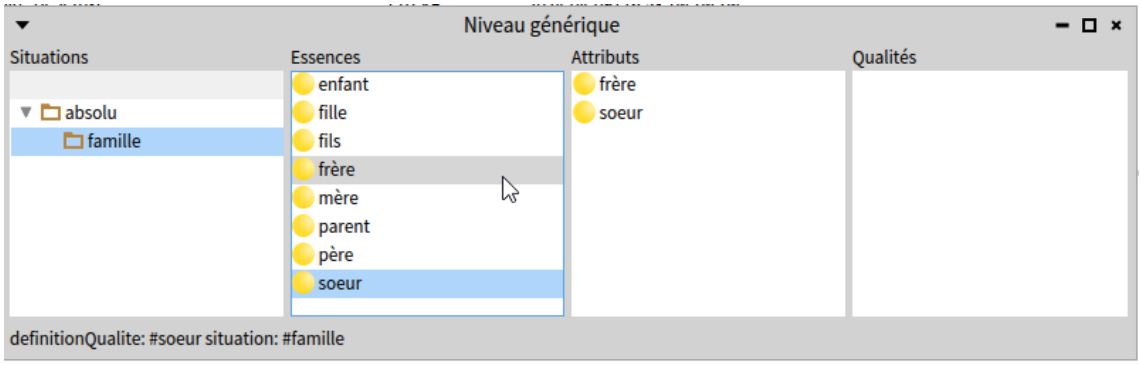
Cet exemple illustre le fait qu'une manière d'être peut être attribut d'elle même.

(Rappelons qu'une essence qui n'est pas une manière d'être ne peut être attribut d'elle même).

Un browser ouvert au niveau générique permet de vérifier par exemple les manières d'être possibles d'une femme :

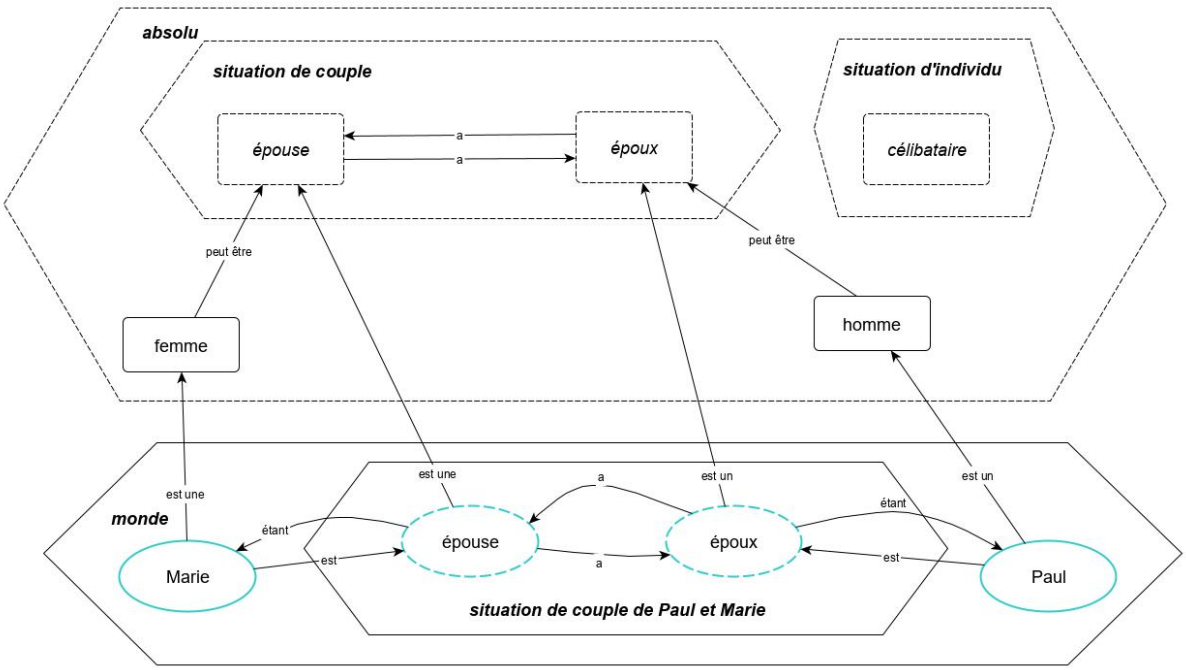


ou les attributs possibles de la manière d'être frère :


Niveau générique
Situations
Essences
Attributs
Qualités
absolu
famille
enfant
fille
fils
frère
mère
parent
père
soeur
frère
soeur
definitionQualite: #soeur situation: #famille

Exemple 14 : sur la relation entre états

Considérons le graphe suivant :




```
1  iceo definition: #femme
2  iceo definition: #homme
3  iceo definitionSituation: 'situation de couple'
4  iceo definitionQualite: #épouse situation: (absolu getSituation: 'situation
       de couple')
5  iceo definitionQualite: #époux situation: (absolu getSituation:
6  'situation de couple')
7  ((absolu getSituation: 'situation de couple') get: #épouse)
       associationQualite: ((absolu getSituation: 'situation de couple')
               get: #époux).
8   femme peutEtre: ((absolu getSituation: 'situation de couple')
       get: #épouse).
9   homme peutEtre: ((absolu getSituation: 'situation de'couple')
       get: #époux).
10   iceo soit: #Marie essence: femme.
11   iceo soit: #Paul essence: homme.
12   iceo soit: 'situation de couple de Paul et Marie' situationGenerique: (
       absolu getSituation: 'situation de couple').
13 (monde get: #Marie) affecteEtat: ((absolu getSituation: 'situation de
       couple') get: #épouse) dansSituation: (monde getSituation: 'situation de
       couple de Paul et Marie').
14  (monde get: #Paul) affecteEtat: ((absolu getSituation: 'situation de
       couple') get: #époux) dansSituation: (monde getSituation: 'situation de
        couple de Paul et Marie').
15 (((monde get: #Paul) getEtat: #époux dansSituation: (monde getSituation: '
       situation de couple de Paul et Marie')) associationEtat: ((monde get: #
       Marie) getEtat: \#épouse dansSituation: (monde getSituation: 'situation
       de couple de Paul et Marie'))).
```

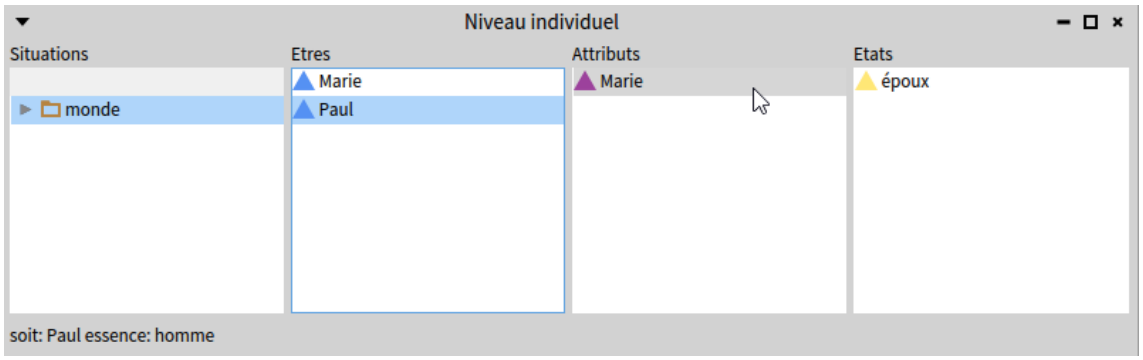
Dans cet exemple, Marie et Paul ne sont pas attributs l'un de l'autre, mais l'expression

```
"    (monde get: #Paul) getEtresAttributsEnTantQue: ((absolu getSituation:
```

```
'situation de couple') get: #époux) "
```

donne pourtant : an OrderedCollection(a femme).

Cet attribut lui est conféré par son état d'époux de Marie (attribut qu'il perdrait en cas de divorce ...)

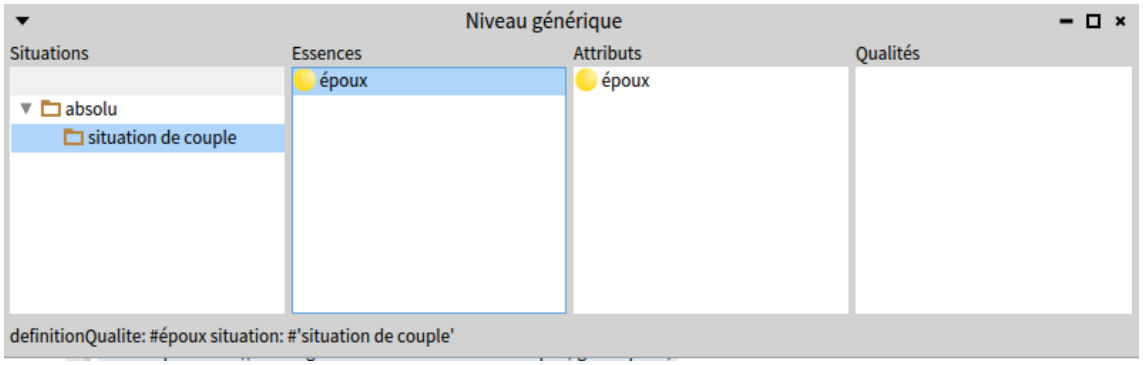


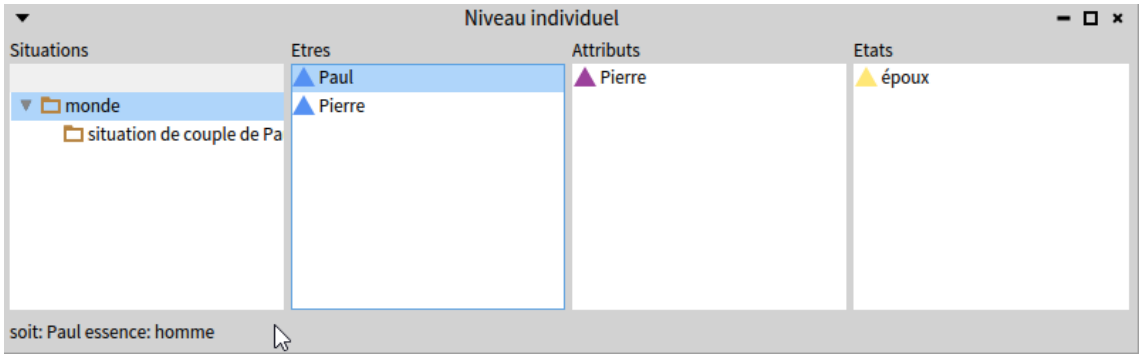
La petite icône violette devant l'attribut Marie indique qu'il ne s'agit pas d'un attribut propre, mais acquis par Paul en sa qualité d'époux.

Exemple 14_2

Cet exemple est une variante du précédent qui représente un couple homosexuel :

```
1  iceo definition: #homme.
2  iceo definitionSituation: 'situation de couple'.
3  iceo definitionQualite: #époux situation: (absolu getSituation: 'situation
       de couple').
4  ((absolu getSituation: 'situation de couple') get: #époux)
       associationQualite: ((absolu getSituation: 'situation de couple') get: #
       époux).
5  homme peutEtre: ((absolu getSituation: 'situation de couple') get: #époux).
6  iceo soit: #Pierre essence: homme.iceo soit: #Paul essence: homme.
7  iceo soit: 'situation de couple de Paul et Pierre' situationGenerique: (
       absolu getSituation: 'situation de couple').
8  (monde get: #Pierre) affecteEtat: ((absolu getSituation: 'situation de
       couple') get: #époux) dansSituation: (monde getSituation: 'situation de
       couple de Paul et Pierre').
9  (monde get: #Paul) affecteEtat: ((absolu getSituation: 'situation de couple
       ') get: #époux) dansSituation: (monde getSituation: 'situation de couple
        de Paul et Pierre').
10 (((monde get: #Paul) getEtat: #époux dansSituation: (monde getSituation: '
       situation de couple de Paul et Pierre')) associationEtat: ((monde get: #
       Pierre) getEtat: #époux dansSituation: (monde getSituation: 'situation
       de couple de Paul et Pierre'))).
```

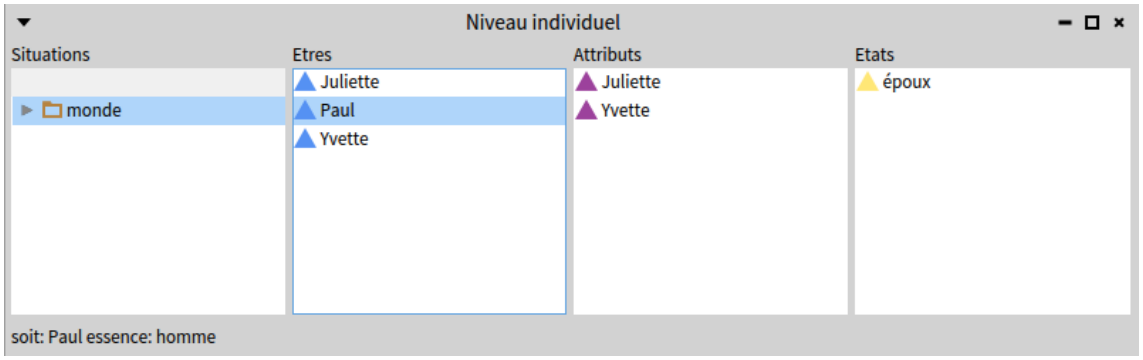




Exemple 14_3

Cet exemple représente le cas d'un homme polygame :

```
1  iceo definition: #femme.
2  iceo definition: #homme.
3  iceo definitionSituation: 'situation de polycouple'.
4  iceo definitionQualite: #épouse situation: (absolu getSituation: 'situation
       de polycouple').
5  iceo definitionQualite: #époux situation: (absolu getSituation: 'situation
       de polycouple').
6  ((absolu getSituation: 'situation de polycouple') get: #épouse)
       associationQualite: ((absolu getSituation: 'situation de polycouple')
       get: #époux).
7  femme peutEtre: ((absolugetSituation: 'situation de polycouple')
       get: #épouse).
8  homme peutEtre: ((absolu getSituation: 'situation de polycouple')
       get: #époux).
9  iceo soit: #Yvette essence: femme.
10 iceo soit: #Juliette essence: femme.
11 iceo soit: #Paul essence: homme.
12 iceo soit: 'situation de polycouple de Paul' situationGenerique: (absolu
       getSituation: 'situation de polycouple').
13 (monde get: #Yvette) affecteEtat: ((absolu getSituation: 'situation de
       polycouple') get: #épouse) dansSituation: (monde getSituation: '
       situation de polycouple de Paul').
14 (monde get: #Juliette) affecteEtat: ((absolu getSituation: 'situation de
       polycouple') get: #épouse) dansSituation: (monde getSituation: '
       situation de polycouple de Paul').
15 (monde get: #Paul) affecteEtat: ((absolu getSituation: 'situation de
       polycouple') get: #époux) dansSituation: (monde getSituation: 'situation
        de polycouple de Paul').
16 (((monde get: #Paul) getEtat: #époux dansSituation: (monde getSituation: '
       situation de polycouple de Paul')) associationEtat: ((monde get: #Yvette
       ) getEtat: #épouse dansSituation: (monde getSituation: 'situation de
       polycouple de Paul'))).
17 (((monde get: #Paul) getEtat: #époux dansSituation: (monde getSituation: '
       situation de polycouple de Paul')) associationEtat: ((monde get: #
       Juliette) getEtat: #épouse dansSituation: (monde getSituation: '
       situation de polycouple de Paul'))).
```

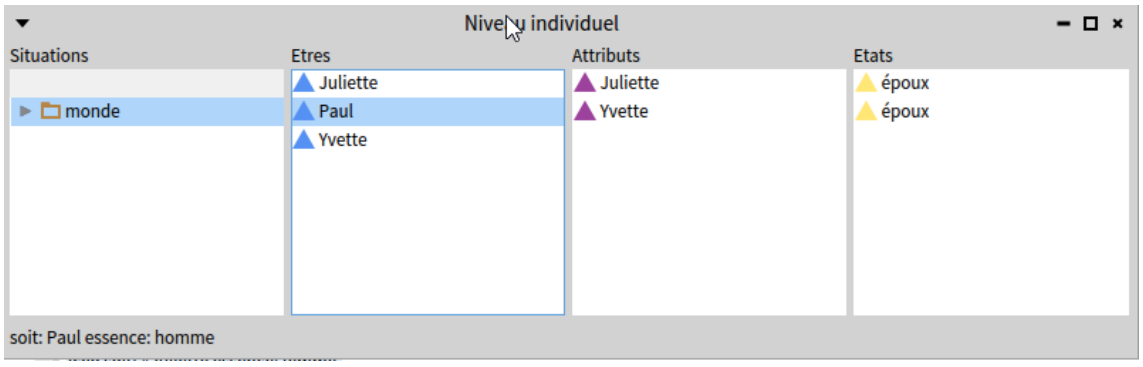


On constate que Paul en tant qu'époux "possède" deux épouses.

Exemple 14_4

Cet exemple est une variante de l'exemple précédent où l'état d'époux de Paul est différent dans sa relation par rapport à Juliette ou Yvette :

```
1  iceo definition: #femme.
2  iceo definition: #homme.
3  iceo definitionSituation: 'situation de polycouple'.
4  iceo definitionQualite: #épouse situation: (absolu getSituation: 'situation
       de polycouple').
5  iceo definitionQualite: #époux situation: (absolu getSituation: 'situation
       de polycouple').
6  ((absolu getSituation: 'situation de polycouple') get: #épouse)
       associationQualite: ((absolu getSituation: 'situation de polycouple')
       get: #époux).
7  femme peutEtre: ((absolu getSituation: 'situation de polycouple')
       get: #épouse).
8  homme peutEtre: ((absolu getSituation: 'situation de polycouple')
       get: #époux).
9  iceo soit: #Yvette essence: femme.
10 iceo soit: #Juliette essence: femme.
11 iceo soit: #Paul essence: homme.
12 iceo soit: 'situation de polycouple de Paul' situationGenerique: (absolu
       getSituation: 'situation de polycouple').
13 (monde get: #Yvette) affecteEtat: ((absolu getSituation: 'situation de
       polycouple') get: #épouse) dansSituation: (monde getSituation: '
       situation de polycouple de Paul').
14 (monde get: #Juliette) affecteEtat: ((absolu getSituation: 'situation de
       polycouple') get: #épouse) dansSituation: (monde getSituation: '
       situation de polycouple de Paul').
15 (monde get: #Paul) affecteEtat: ((absolu getSituation: 'situation de
       polycouple') get: #époux) dansSituation: (monde getSituation: 'situation
        de polycouple de Paul').
16 (monde get: #Paul) affecteEtat: ((absolu getSituation: 'situation de
       polycouple') get: #époux) dansSituation: (monde getSituation: 'situation
        de polycouple de Paul').
17 ((((monde get: #Paul) getEtats: #époux dansSituation: (monde getSituation:
       'situation de polycouple de Paul')) at: 1) associationEtat: ((monde get:
        #Yvette) getEtat: #épouse dansSituation: (monde getSituation: '
       situation de polycouple de Paul'))).
18 ((((monde get: #Paul) getEtats: #époux dansSituation: (monde getSituation:
       'situation de polycouple de Paul')) at: 2) associationEtat: ((monde get:
        #Juliette) getEtat: #épouse dansSituation: (monde getSituation: '
       situation de polycouple de Paul'))).
```



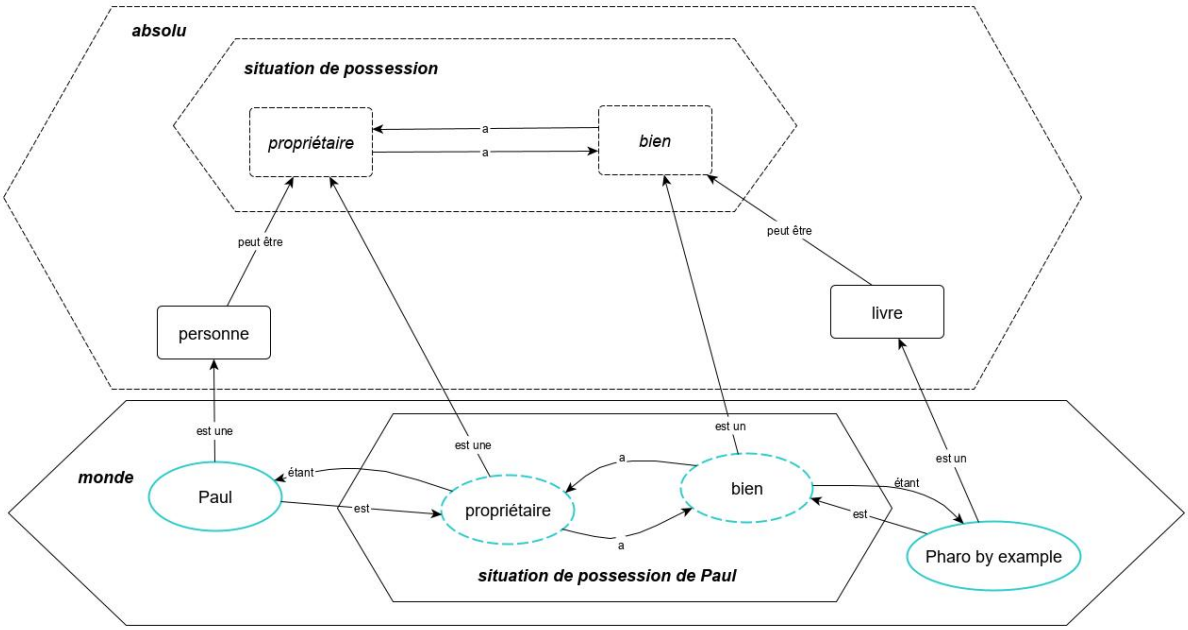
Cet exemple est plus intéressant que le précédent car, étant donné qu'un état peut être

dans un état, il permettrait de préciser que Paul est un mauvais époux pour Yvette et un bon époux pour Juliette !

Exemple 14_5

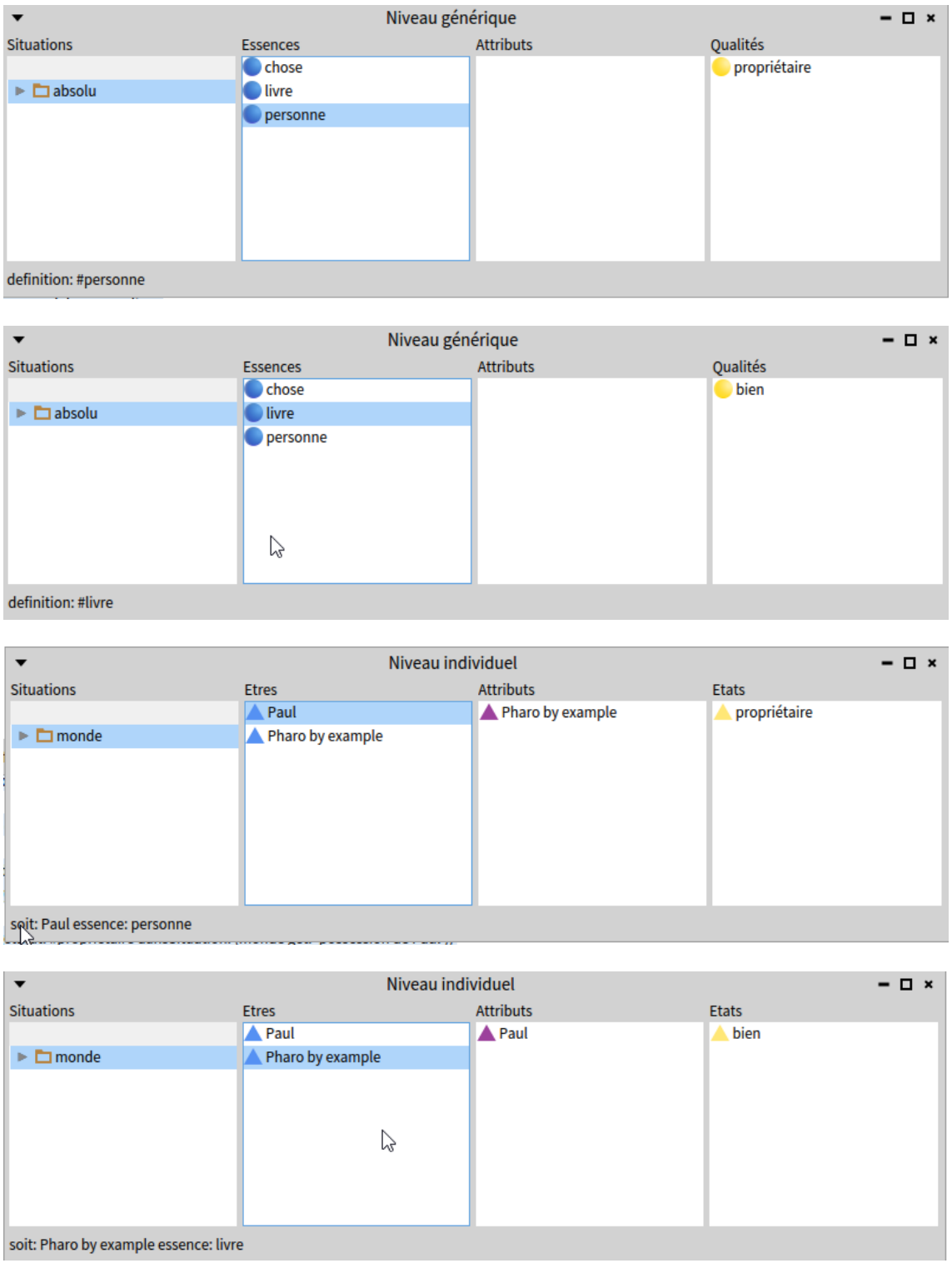
Cet exemple illustre la relation possible entre deux êtres qui ne soit pas une relation d'inclusion.

Il représente le fait qu'une personne puisse être propriétaire d'un livre :




```
1  iceo definition: #personne.
2  iceo definition: #livre.
3  iceo definitionSituation: #possession.
4  iceo definitionQualite: #propriétaire situation: possession.
5  iceo definitionQualite: #bien situation: possession.
6  (possession get: #propriétaire) associationQualite: (possession get: #bien)
       .
7  personne peutEtre: (possession get: #propriétaire).livre peutEtre: (
       possession get: #bien).
8  iceo soit: #Paul essence: personne.
9  iceo soit: 'Pharo by example' essence: livre.
10 iceo soit: 'possession de Paul' situationGenerique: possession.
11 (monde get: #Paul) affecteEtat: (possession get: #propriétaire)
       dansSituation: (monde getSituation: 'possession de Paul').
12 (monde get: 'Pharo by example') affecteEtat: (possession get: #bien)
       dansSituation: (monde get: 'possession de Paul') .
13 ((monde get: #Paul) getEtat: #propriétaire dansSituation: (monde get: '
       possession de Paul')) associationEtat: ((monde get: 'Pharo by example')
       getEtat: #bien dansSituation: (monde get: 'possession de Paul')).
```

Le résultat de cette construction apparait dans les fenêtres suivantes :



On voit que l'attribut 'Phoro by example' de Paul lui est conféré par son état de propriétaire (icône violette).

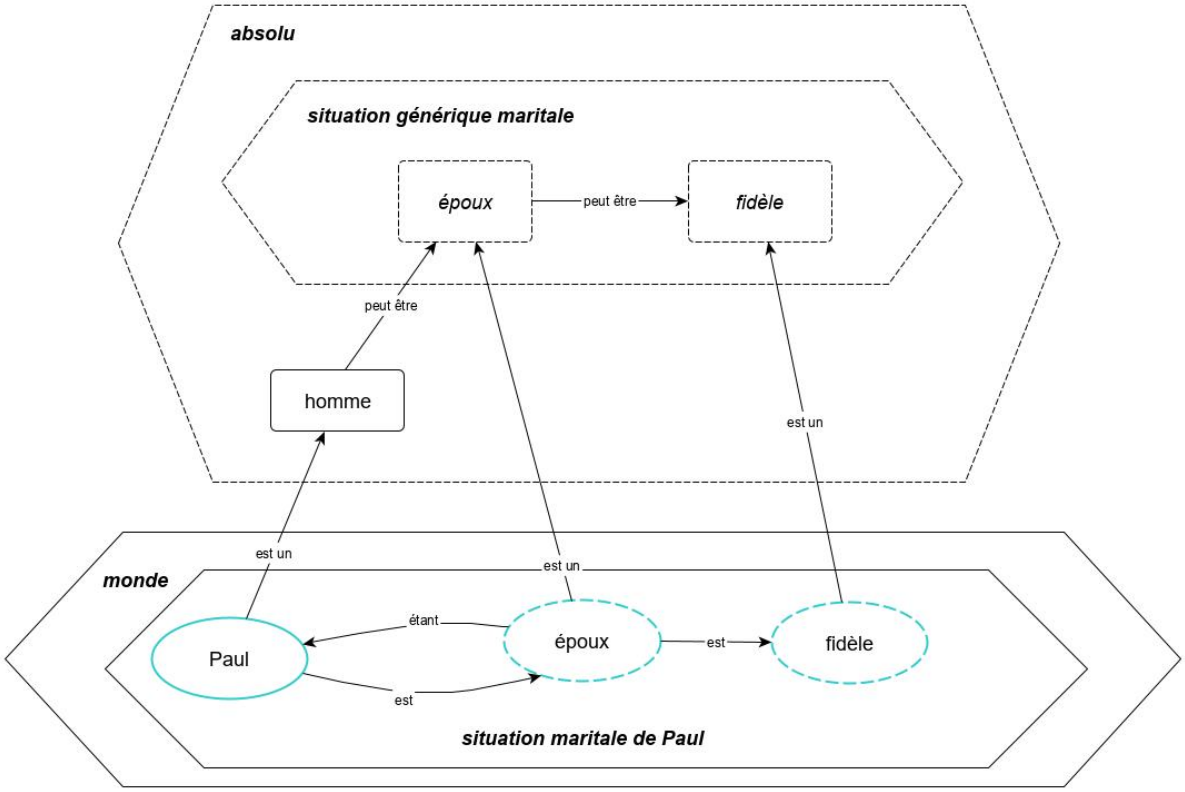
Le fait de dire que le propriétaire d'un livre est une personne (comme nous l'avons fait dans l'exemple 7) est un raccourci de langage.

Bien sûr, un livre ne rentre pas dans la composition d'une personne, et réciproquement !

La même logique s'appliquerait pour dire que Paul a un crayon, une veste, ...

Exemple 15 : sur la notion d'état dans un état

Considérons le graphe suivant :

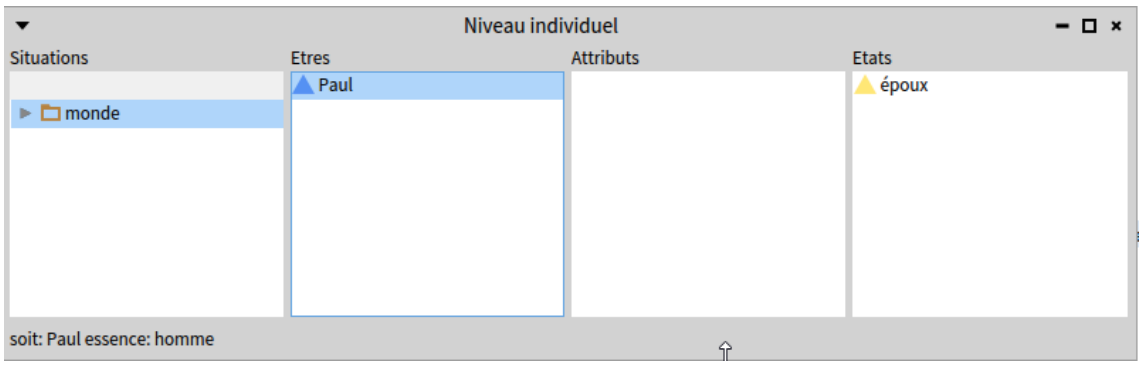


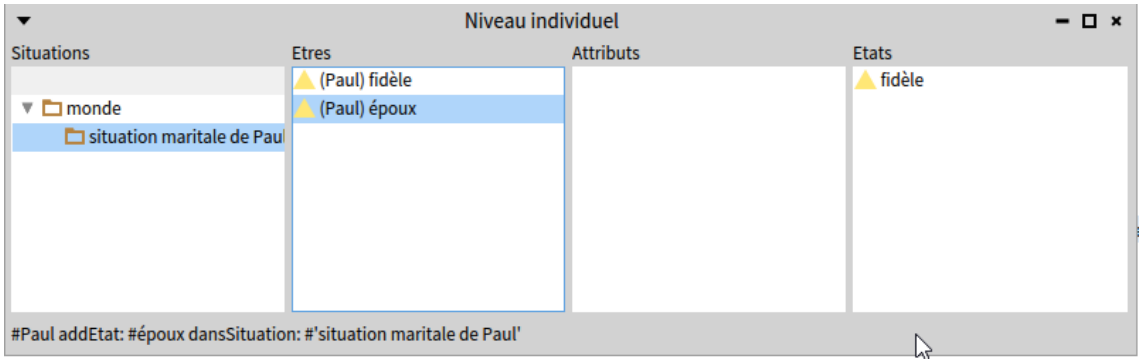

```
1  iceo definition: #homme.
2  iceo definitionSituation: 'situation générique maritale'.
3  iceo definitionQualite: #époux situation:(absolu getSituation: 'situation g
       énérique maritale').
4  iceo definitionQualite: #fidèle situation: (absolu getSituation:     '
       situation générique maritale').
5  homme peutEtre: ((absolu getSituation: 'situation générique maritale') get:
        #époux).
6  iceo soit: 'situation maritale de Paul' situationGenerique: (absolu
       getSituation: 'situation générique maritale').
7  iceo soit: #Paul essence: homme.
8  (monde get: #Paul) affecteEtat: ((absolu getSituation: 'situation générique
        maritale') get: #époux) dansSituation: (monde get: 'situation maritale
       de Paul').
9  ((monde get: #Paul) getEtat: #époux dansSituation: (monde get: 'situation
       maritale de Paul')) affecteEtat: ((absolu getSituation: 'situation géné
       rique maritale') get: #fidèle) dansSituation: (monde get: 'situation
       maritale dePaul').
```

L'expression " `((monde get: #Paul) getEtat: #époux dansSituation: (monde get: 'situation maritale de Paul')) getEtats` "

donne : an OrderedCollection(a fidèle)

Ceci est visualisé dans les fenêtres suivantes :



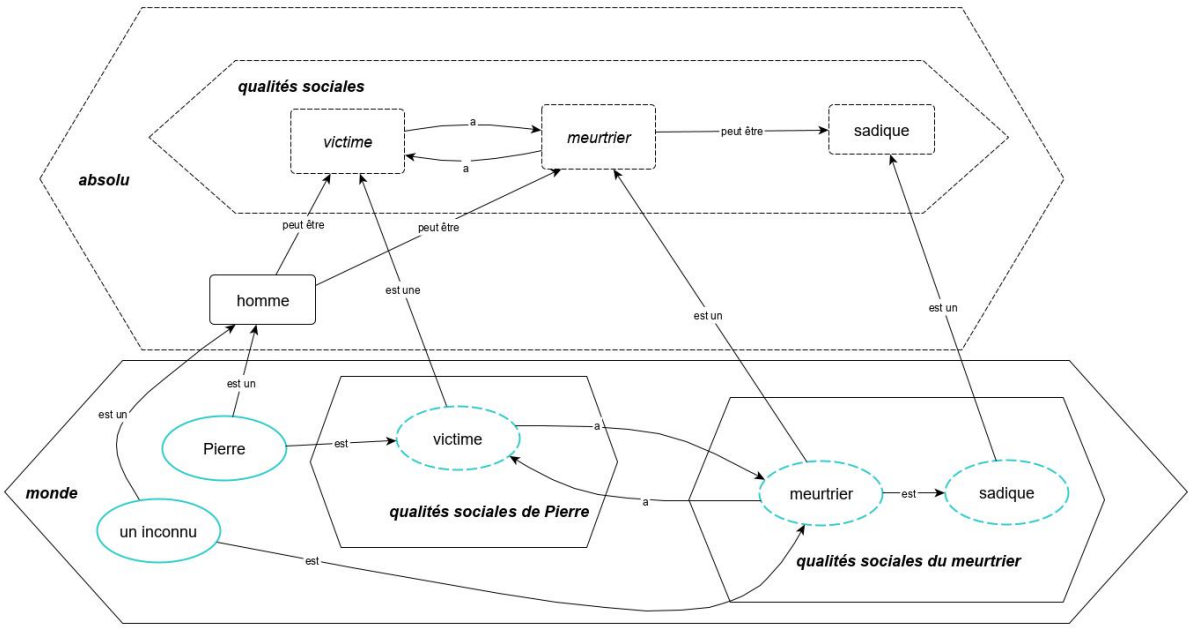


Noter que l'expression " `((monde get: \#Paul) getEtat: #fidèle dansSituation: (monde get: 'situation maritale de Paul'))` "

ne donnerait rien, car si Paul est fidèle en tant qu'époux, il ne l'est pas forcément dans toutes les situations !

Exemple 16 : sur la notion d'être inconnu

Considérons le graphe suivant :



Il correspond à la représentation de la phrase : "Le meurtrier de Pierre est sadique ".

Cette affirmation n'implique pas la connaissance de l'identité du meurtrier par celui qui la prononce. Nous admettrons que cette phrase a un sens, bien que le meurtrier de Pierre puisse être inconnu; elle peut être formulée en voyant simplement l'état épouvantable de la victime Pierre.

Sa représentation dans ICEO est la suivante :

```
1  iceo definition: #personne
2  iceo definition: #homme genus: personne
3  iceo definitionSituation: 'qualités sociales'
4  iceo definitionQualite: #meurtrier situation: (absolu getSituation: #'
       qualités sociales')
5  iceo definitionQualite: #victime situation: (absolu getSituation: #'qualité
       s sociales')
6  iceo definitionQualite: #sadique situation: (absolu getSituation: #'qualité
       s sociales')
7  homme peutEtre: ((absolu getSituation: #'qualités sociales') get: #
       meurtrier).
8  homme peutEtre: ((absolu getSituation: #'qualités sociales') get: #victime)
       .
9  ((absolu getSituation: #'qualités sociales') get: #victime)
       associationQualite: ((absolu getSituation: #'qualités sociales') get: #
       meurtrier).
10 ((absolu getSituation: #'qualités sociales') get: #meurtrier) peutEtre: ((
       absolu getSituation: #'qualités sociales') get: #sadique).
11 iceo soit: #Pierre essence: homme.
12 iceo soit: 'qualités sociales de Pierre' situationGenerique: (absolu
       getSituation: #'qualités sociales').
13 (monde get: #Pierre) affecteEtat: ((absolu getSituation: #'qualités
       sociales') get: #victime) dansSituation: (monde get: #'qualités sociales
        de Pierre' ).
```

```
14  iceo soit: 'qualités sociales du meurtrier' situationGenerique: (absolu
        getSituation: #'qualités sociales').
15  " Création d'un état dont l'étant est inconnu"
16  iceo soitEtat: #meurtrier essence: ((absolu getSituation: #'qualités
        sociales')    get: #meurtrier) situationIndividuelle: (monde get: #'
        qualités sociales du meurtrier' ).
17  ((monde get: #'qualités sociales du meurtrier' ) get: #meurtrier)
        affecteEtat: ((absolu getSituation: #'qualités sociales') get: #sadique)
         dansSituation:    (monde get: #'qualités sociales du meurtrier' ).
18  ((monde get: #'qualités sociales du meurtrier' ) get: #meurtrier)
        associationEtat:     ((monde get: #'qualités sociales de Pierre' ) get:
        #victime).
```

`((monde get: 'qualités sociales du meurtrier' ) get: #meurtrier) getEtats"`

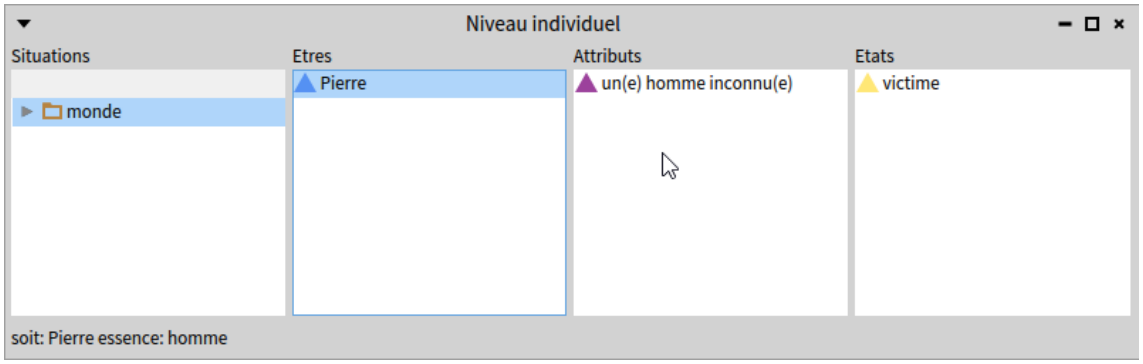
donne : an OrderedCollection(a sadique)

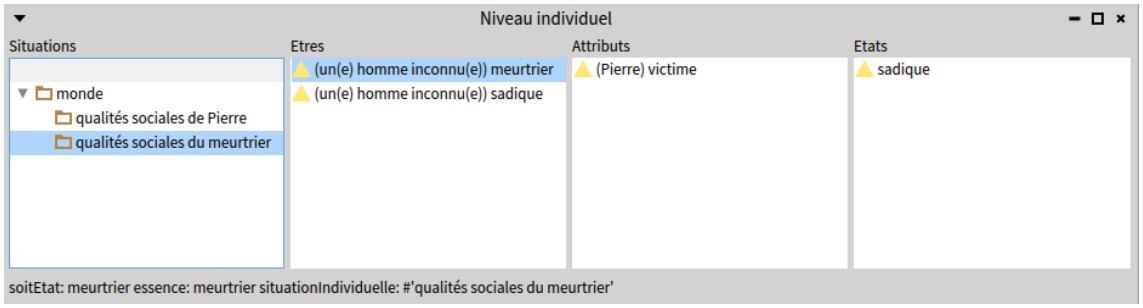
" `(monde get: #Pierre) getEtats` " donne : an OrderedCollection(a victime)

Enfin, l'expression " `((monde get: #'qualités sociales du meurtrier') get: #meurtrier) getEtant getNom` "

donne : 'un(e) homme inconnu(e)'

Ceci est visualisé dans les fenêtres suivantes :

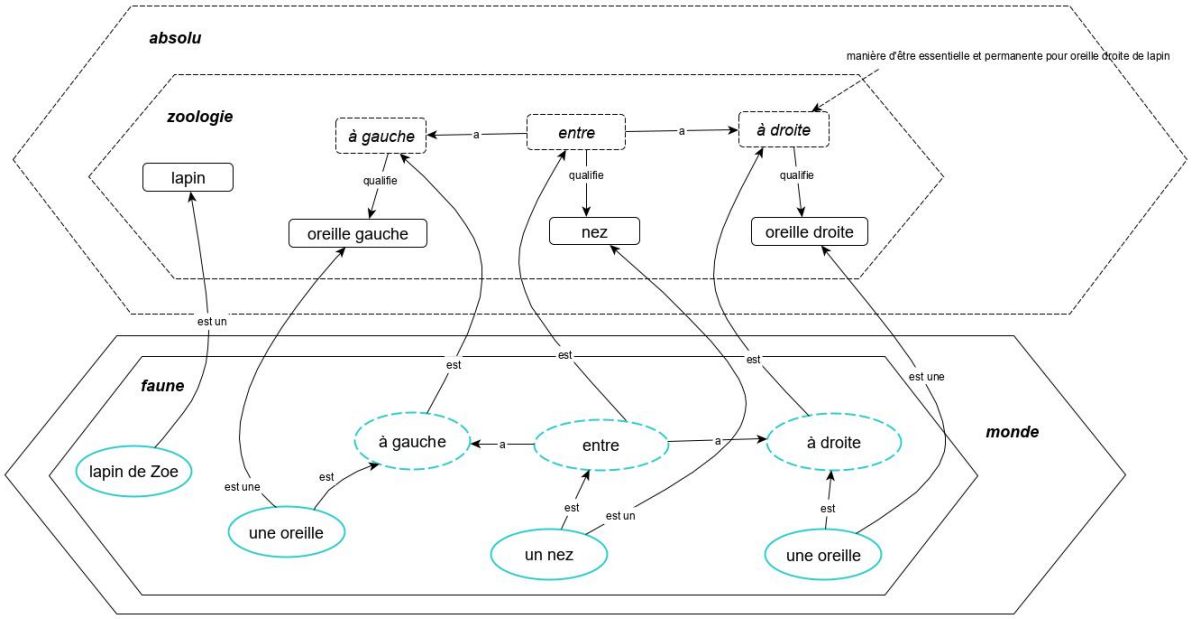




Si plusieurs essences avaient la qualité de meurtrier, c'est leur premier genus commun qui aurait été choisi comme essence du meurtrier. Si ce genus était l'essence chose, le meurtrier aurait été simplement identifié comme étant "quelque chose".

Exemple 17 : sur la notion de contrainte structurelle interne

Le graphe suivant exprime que le nez d'un lapin doit se situer entre ses deux oreilles :



(Pour ne pas complexifier l'aspect visuel de ce diagramme, le fait que le nez et les oreilles sont des attributs du lapin n'a pas été représenté).

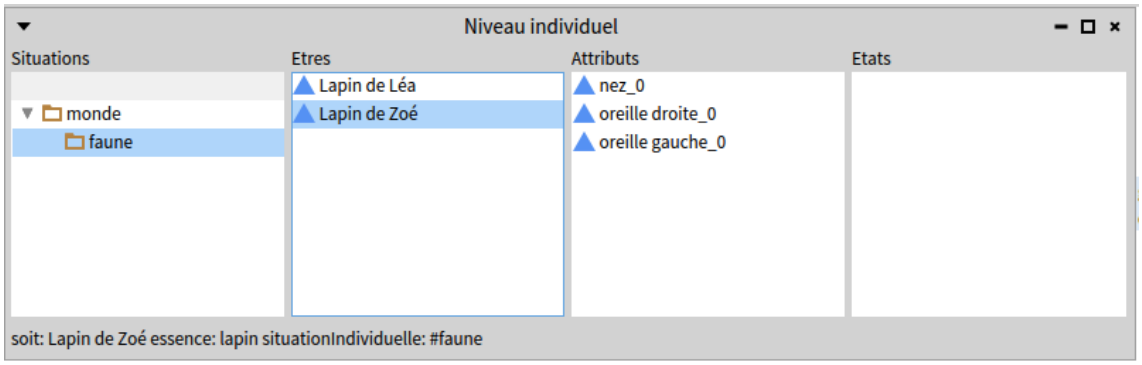
Ce graphe exprime que les manières d'être "à gauche", "entre" et "à droite" sont essentielles et permanentes pour les essences oreille gauche, nez et oreille droite de lapin.

La représentation dans ICEO est la suivante :

```
1  iceo definitionSituation: #zoologie
2  iceo definition: #lapin situation: zoologie
3  iceo definitionAttribut: #nez de: (zoologie get: #lapin) cardinalite: 1
4  iceo definitionAttribut: 'oreille gauche' de: (zoologie get: #lapin)
       cardinalite: 1
5  iceo definitionAttribut: 'oreille droite' de: (zoologie get: #lapin)
       cardinalite: 1
6  iceo definitionQualiteEssentielle: 'entre' genus: ((absolu getSituation:
         'contrainte de position relative') get: 'entre') pour: ((zoologie get
       : #lapin) getEssenceAttribut: #nez) effectivite: #permanente
7  iceo definitionQualiteEssentielle: 'à gauche' genus: ((absolu getSituation:
       'contrainte de position relative') get: 'à gauche') pour: ((zoologie
      get: #lapin) getEssenceAttribut: 'oreille gauche') effectivite: #
      permanente
8  iceo definitionQualiteEssentielle: 'à droite' genus: ((absolu getSituation:
       'contrainte de position relative') get: 'à droite') pour: ((zoologie
      get: #lapin) getEssenceAttribut: 'oreille droite') effectivite: #
      permanente
9  (((zoologie get: #lapin) getEssenceAttribut: #nez) getQualite: 'entre')
       associationQualite: (((zoologie get: #lapin) getEssenceAttribut: '
       oreille gauche') getQualite: 'à gauche').
10 (((zoologie get: #lapin) getEssenceAttribut: #nez) getQualite:'entre')
11  associationQualite: (((zoologie get: #lapin) getEssenceAttribut: 'oreille
```

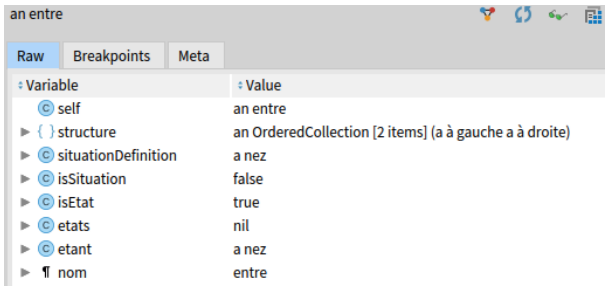
```
        droite') getQualite: 'à droite').
12  iceo soit: #faune situationGenerique: zoologie
13  iceo soit: 'Lapin de Zoé' essence: (zoologie get: #lapin)
        situationIndividuelle: (monde get: #faune)
14  (((monde get: 'Lapin de Zoé') getEtreAttribut: #nez) getEtat: #entre)
        associationEtat: (((monde get: 'Lapin de Zoé') getEtreAttribut: 'oreille
        gauche') getEtat: 'à gauche')
15  (((monde get: 'Lapin de Zoé') getEtreAttribut: #nez) getEtat: #entre)
        associationEtat: (((monde get: 'Lapin de Zoé') getEtreAttribut: 'oreille
        droite') getEtat: 'à droite')
```

Voici une visualisation du lapin de Zoé :

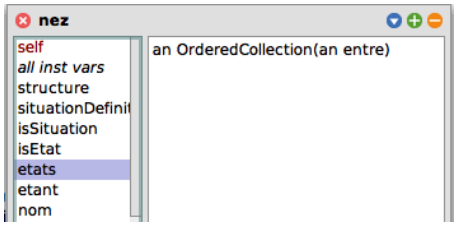


Le nez du lapin de Zoé à été automatiquement instancié, comme ses oreilles, car leur cardinalité au niveau de l'essence lapin est de 1.

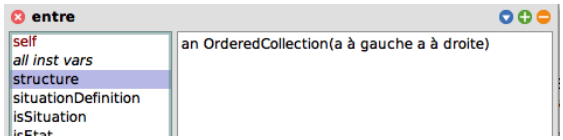
L'inspection de " `((monde get: 'Lapin de Zoé') getEtreAttribut: \#nez) getEtats at: 1` " donne :



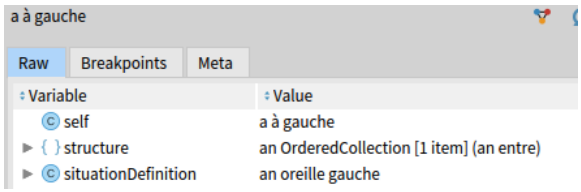
Faisons l'inspection du nez :



L'inspection de l'état "entre" donne :



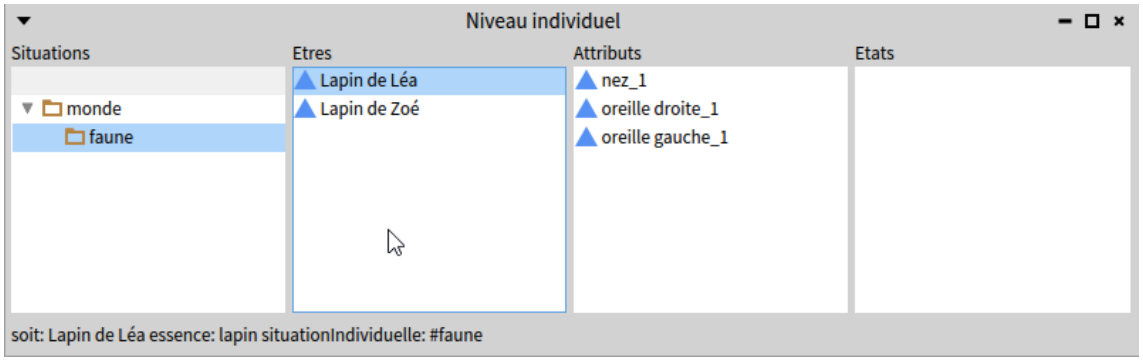
L'inspection de l'état "à gauche" donne :



On constate qu'il s'agit bien de l'état de l'oreille gauche du lapin de Zoé.

Considérons maintenant un lapin nommé Léa :

```
1 iceo soit: 'Lapin de Léa' essence: (zoologie get: #lapin)
    situationIndividuelle: (monde get: #faune)
2 (((monde get: 'Lapin de Léa') getEtreAttribut: #nez) getEtat: #entre)
    associationEtat: (((monde get: 'Lapin de Léa') getEtreAttribut: 'oreille
     gauche') getEtat: #'à gauche')
3 (((monde get: 'Lapin de Léa') getEtreAttribut: #nez) getEtat: #entre)
    associationEtat: (((monde get: 'Lapin de Léa') getEtreAttribut: 'oreille
     droite') getEtat: #'à droite')
```



On constate que les oreilles des deux lapins sont différentes, ce qui est logique.

Mais qu'en est-il de leurs états ? Faisons un test d'identité en évaluant l'expression

```
" (((monde get: 'Lapin de Zoé') getEtreAttribut: \#'oreille gauche')
getEtats at: 1) == ((monde get: 'Lapin de Léa') getEtreAttribut: \#'oreille
gauche') getEtats at: 1) "
```

Le résultat est : false

Ce résultat est tout-à-fait dans la logique d'ICEO.

En effet, il est important de distinguer l'état "à gauche" de l'oreille gauche du lapin de Léa de l'état "à gauche" de l'oreille gauche du lapin de Zoé.

Sachant que, comme nous l'avons vu, un état peut lui même avoir des états, il serait possible par exemple que l'un soit "un peu plus à gauche" que l'autre.

Exemple 18 : sur la notion de famille

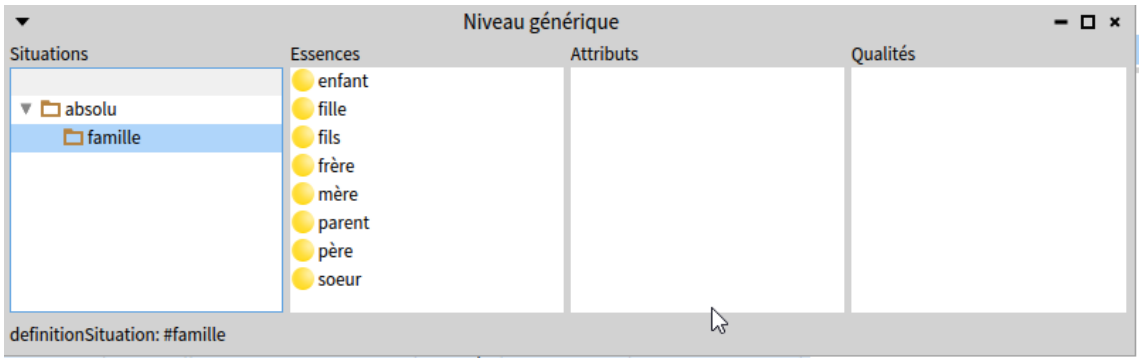
Cet exemple repart de l'exemple 13 (qui peut être réévalué en bloc) sur la subsomption des manières d'être pour représenter le famille Gonthier où les parents Charles et Hermeline possèdent deux filles Capucine et Juliette et deux fils Martin et Jean :

```
1  iceo soit: #Charles essence: homme.
2  iceo soit: #Hermeline essence: femme.
3  iceo soit: #Martin essence: homme.
4  iceo soit: #Jean essence: homme.
5  iceo soit: #Capucine essence: femme.
6  iceo soit: #Juliette essence: femme.
7  (monde get: #Charles) affecteEtat: ((absolu getSituation: 'famille\
       textquotesingle) get: #père) dansSituation: (monde get: #Gonthier).
8  (monde get: #Hermeline) affecteEtat: ((absolu getSituation: 'famille\
       textquotesingle) get: #mère) dansSituation: (monde get: #Gonthier).
9  (monde get: #Martin) affecteEtat: ((absolu getSituation: 'famille\
       textquotesingle) get: #fils) dansSituation: (monde get: #Gonthier).
10 ((monde get: #Martin) getEtat: #fils dansSituation: (monde get: #Gonthier))
        affecteEtat: ((absolu getSituation: #famille) get: #frère)
       dansSituation: (monde get: #Gonthier).
11 (monde get: #Jean) affecteEtat: ((absolu getSituation: 'famille\
       textquotesingle) get: #fils) dansSituation: (monde get: #Gonthier).
12 ((monde get: #Jean) getEtat: #fils dansSituation: (monde get: #Gonthier))
       affecteEtat: ((absolu getSituation: #famille) get: #frère) dansSituation
       : (monde get: #Gonthier).
13 (monde get: #Capucine) affecteEtat: ((absolu getSituation: 'famille\
       textquotesingle) get: #fille) dansSituation: (monde get: #Gonthier).
14 ((monde get: #Capucine) getEtat: #fille dansSituation: (monde get: #
       Gonthier))
15  affecteEtat: ((absolu getSituation: #famille) get: #soeur) dansSituation:
        (monde get: #Gonthier).
16 (monde get: #Juliette) affecteEtat: ((absolu getSituation: 'famille\
       textquotesingle) get: #fille) dansSituation: (monde get: #Gonthier).
17 ((monde get: #Juliette) getEtat: #fille dansSituation: (monde get: #
       Gonthier))
18  affecteEtat: ((absolu getSituation: #famille) get: #soeur) dansSituation:
        (monde get: #Gonthier).
19 ((monde get: #Charles) getEtat: #père dansSituation: (monde get: #Gonthier)
       ) associationEtat: ((monde get: #Capucine) getEtat: #fille dansSituation
       : (monde get: #Gonthier)).
20 ((monde get: #Charles) getEtat: #père dansSituation: (monde get: #Gonthier)
       ) associationEtat: ((monde get: #Martin) getEtat: #fils dansSituation: (
       monde get: #Gonthier)).
21 ((monde get: #Charles) getEtat: #père dansSituation: (monde get: #Gonthier)
       ) associationEtat: ((monde get: #Jean) getEtat: #fils dansSituation: (
       monde get: #Gonthier)).
22 ((monde get: #Charles) getEtat: #père dansSituation: (monde get: #Gonthier)
       ) associationEtat: ((monde get: #Juliette) getEtat: #fille dansSituation
       : (monde get: #Gonthier)) .
23 ((monde get: #Hermeline) getEtat: #mère dansSituation: (monde get: #
       Gonthier))  associationEtat: ((monde get: #Capucine) getEtat: #fille
       dansSituation: (monde get: #Gonthier)).
24 ((monde get: #Hermeline) getEtat: #mère dansSituation: (monde get: #
       Gonthier))
25  associationEtat: ((monde get: #Martin) getEtat: #fils dansSituation: (
        monde get: #Gonthier)).
26 ((monde get: #Hermeline) getEtat: #mère dansSituation: (monde get: #
       Gonthier)) associationEtat: ((monde get: #Jean) getEtat: #fils
```

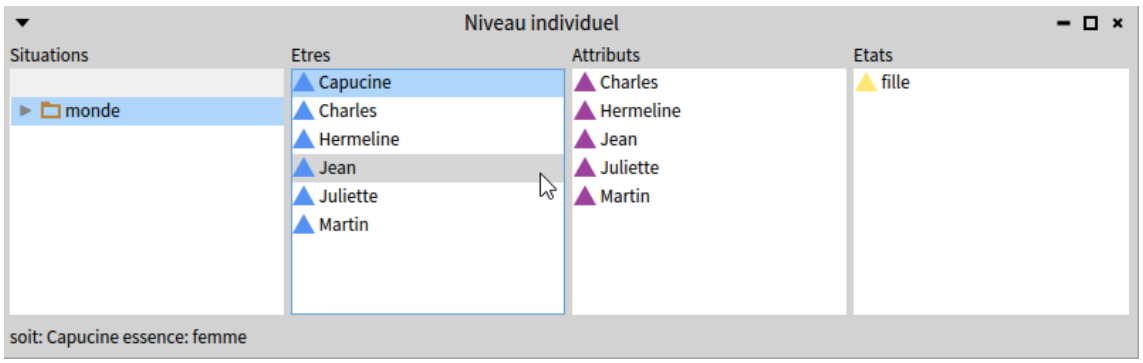
```
    dansSituation: (monde get: #Gonthier)).
27  ((monde get: #Hermeline) getEtat: #mère dansSituation: (monde get: #
    Gonthier)) associationEtat: ((monde get: #Juliette) getEtat: #fille
    dansSituation: (monde get: #Gonthier)) .
28  (((monde get: #Capucine) getEtat: #fille dansSituation: (monde get: #
    Gonthier)) getEtat: #soeur
29   dansSituation: (monde get: #Gonthier)) associationEtat: (((monde get: #
     Martin) getEtat: #fils
30   dansSituation: (monde get: #Gonthier)) getEtat: #frère dansSituation: (
     monde get: #Gonthier)) .
31  (((monde get: #Capucine) getEtat: #fille dansSituation: (monde get: #
    Gonthier)) getEtat: #soeur
32   dansSituation: (monde get: #Gonthier)) associationEtat: (((monde get: #
     Jean) getEtat: #fils dansSituation: (monde get: #Gonthier)) getEtat: #
     frère dansSituation: (monde get: #Gonthier)).
33  (((monde get: #Capucine) getEtat: #fille dansSituation: (monde get: #
    Gonthier)) getEtat: #soeur dansSituation: (monde get: #Gonthier))
    associationEtat: (((monde get: #Juliette) getEtat: #fille dansSituation:
     (monde get: #Gonthier)) getEtat: #soeur dansSituation: (monde get: #
    Gonthier)).
34  (((monde get: #Juliette) getEtat: #fille dansSituation: (monde get: #
    Gonthier)) getEtat: #soeur dansSituation: (monde get: #Gonthier))
    associationEtat: (((monde get: #Jean) getEtat: #fils dansSituation: (
    monde get: #Gonthier)) getEtat: #frère dansSituation:
35  (monde get: #Gonthier)).
36  (((monde get: #Juliette) getEtat: #fille dansSituation: (monde get: #
    Gonthier)) getEtat: #soeur dansSituation: (monde get: #Gonthier)) ,
    associationEtat: (((monde get: #Martin) getEtat: #fils dansSituation: (
    monde get: #Gonthier)) getEtat: #frère dansSituation: (monde get: #
    Gonthier)).
37  (((monde get: #Martin) getEtat: #fils dansSituation: (monde get: #Gonthier)
    ) getEtat: #frère dansSituation: (monde get: #Gonthier)) associationEtat
    : (((monde get: #Jean) getEtat: #fils dansSituation: (monde get: #
    Gonthier)) getEtat: #frère dansSituation: (monde get: #Gonthier)).
```

Au niveau générique, rien n'a changé par rapport à l'exemple 13.

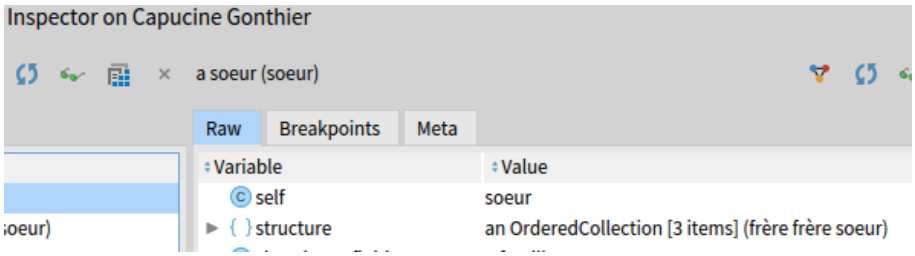
Par exemple :



Au niveau individuel, Capucine est une fille :



En tant que fille, elle est soeur de Juliette, Jean et Martin :



ICEO étant implanté en Smalltalk, il est possible de mixer du code ICEO avec du code Smalltalk.

Nous en avons profité pour surcharger la méthode "printOn: aStream" de la classe Object de Smalltalk au niveau de l'essence chose, pour afficher le nom des membres d'une famille avec leur nom de famille :

```
chose compile: ' printOn: aStream
self isEtat
        ifTrue: [self getEtats
                detect: [:e | e getSituationDefinition class getNom == #famille]
                ifFound: [:x | aStream nextPutAll: self getEtant getNom , '' '' , x getSituationDefinition getNom]
                ifNone: [aStream nextPutAll: self getNom]]
        ifFalse: [ self getEtats
                detect: [:e | e getSituationDefinition class getNom == #famille]
                ifFound: [:x | aStream nextPutAll: self getNom , '' '' , x getSituationDefinition getNom]
                ifNone: [aStream nextPutAll: self getNom]]'
```

Ainsi, voici les réponses données par diverses expressions :

`((monde get: #Charles) getEtresAttributsQuiSont: ((absolu getSituation: #famille) get: #enfant))` donne : an OrderedCollection(Capucine Gonthier Martin Gonthier Jean Gonthier Juliette Gonthier)

`((monde get: #Charles) getEtresAttributsQuiSont: ((absolu getSituation: #famille) get: #fille))` donne : an OrderedCollection(Capucine Gonthier Juliette Gonthier)

`((monde get: #Capucine) getEtresAttributsQuiSont: ((absolu getSituation: #famille) get: #frère))` donne : an OrderedCollection(Martin Gonthier Jean

Gonthier)

`((monde get: #Martin) getEtresAttributsQuiSont: ((absolu getSituation: #famille) get: #soeur))` donne : an OrderedCollection(Capucine Gonthier Juliette Gonthier)

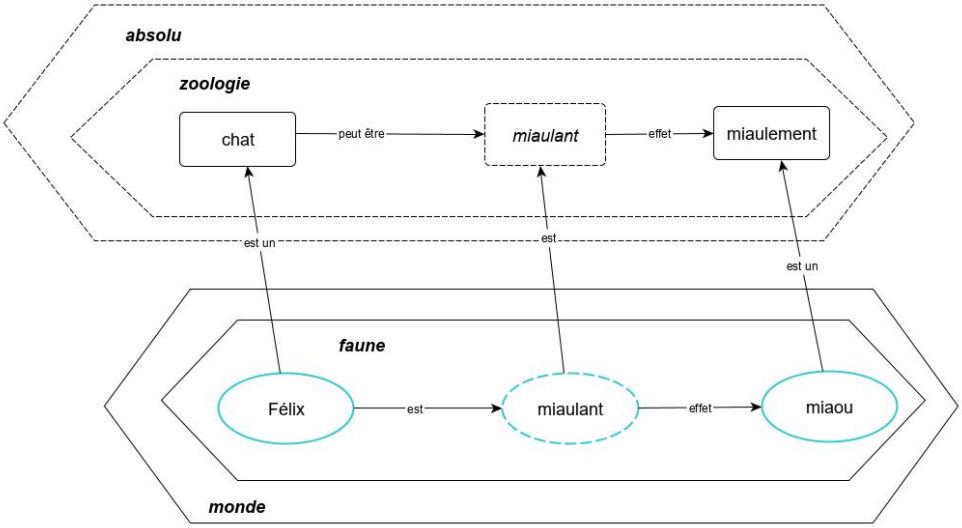
`((monde get: #Martin) getEtresAttributsQuiSont: ((absolu getSituation: #famille) get: #frère))` donne : an OrderedCollection(Jean Gonthier)

Noter que dans cet exemple Capucine possède, en tant que soeur, une soeur et deux frères, mais il ne serait pas possible de préciser qu'elle est un gentille soeur pour Juliette et une chipie par rapport à Martin.

Pour cela, il faudrait procéder comme nous l'avons fait dans l'exemple 14_4 pour différencier les deux états d'époux de Paul.

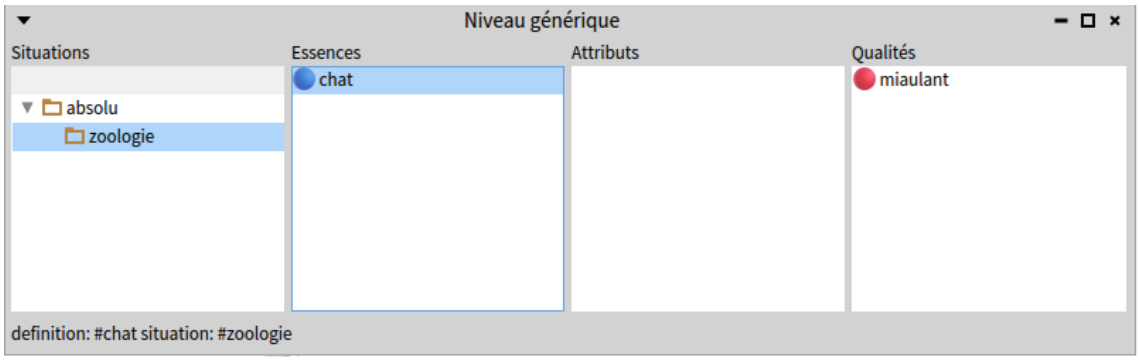
Exemple 19 : sur la notion d'action

Considérons le graphe suivant :




```
1 iceo definitionSituation: #zoologie.
2 iceo definition: #chat situation: zoologie.
3 iceo definitionQualiteEssentielle: #miaulant pour: (zoologie
4 get: #chat) effectivite: #intermittente.
5 iceo definitionAttribut:#miaulement de: ((zoologie get: #chat) getQualite:
    #miaulant).
```

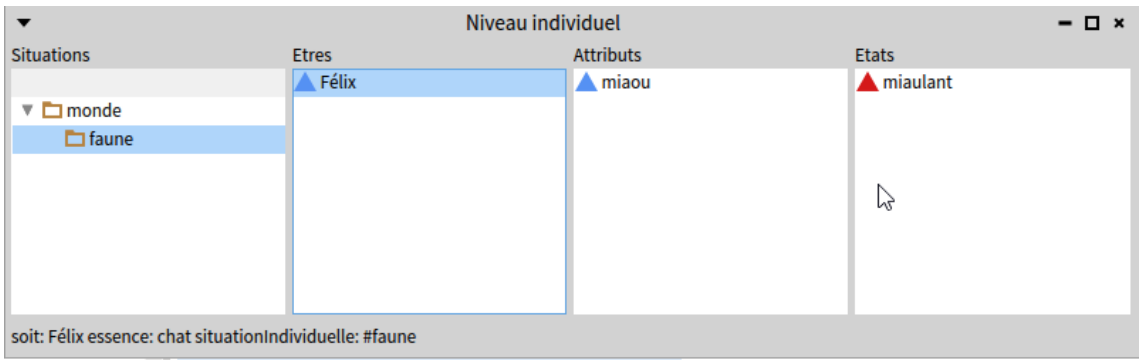
Dans cet exemple, la qualité "miaulant" de chat est essentielle (le chat peut miauler par essence)



mais son effectivité est intermittente (un chat ne miaule pas sans arrêt).

Il est donc nécessaire de préciser que Félix est ici en train de miauler.

```
1 iceo soit: #faune situationGenerique: zoologie
2 iceo soit: #Félix essence: (zoologie get: #chat)
3 ((monde get: #faune) get: #Félix) affecteEtatEssentiel: ((zoologie get: #
    chat) getQualite: #miaulant)
4 (((monde get: #faune) get: #Félix) getEtat: #miaulant) attributionEtre: #
    miaou essence: ((((zoologie get: #chat) getQualite: #miaulant))
    getEssenceAttribut: #miaulement)
```



Supposons que la méthode Smalltalk suivante soit associée à la manière d'être miaulant :

```
((zoologie get: #chat) getQualite: #miaulant) compile: 'miaule .
    ^(self getEtresAttributs asOrderedCollection at: 1) getNom'.
```

L'envoi du message "miaule" à l'état miaulant de Félix par :

```
(((monde get: #faune) get: #Félix) getEtat: #miaulant) miaule
```

donne : Miaou

Supposons que Maya soit une chatte dont le miaulement produit "meow" :

```
1 iceo soit: #Maya essence: (zoologie get: #chat)
2 ((monde get: #faune) get: #Maya) affecteEtatEssentiel: ((zoologie get: #
      chat) getQualite: #miaulant)
3 (((monde get: #faune) get: #Maya) getEtat: #miaulant) attributionEtre: #
      meow essence: ((((zoologie get: #chat) getQualite: #miaulant))
      getEssenceAttribut: #miaulement)
```

L'envoi du message miaule à l'état miaulant de Maya par :

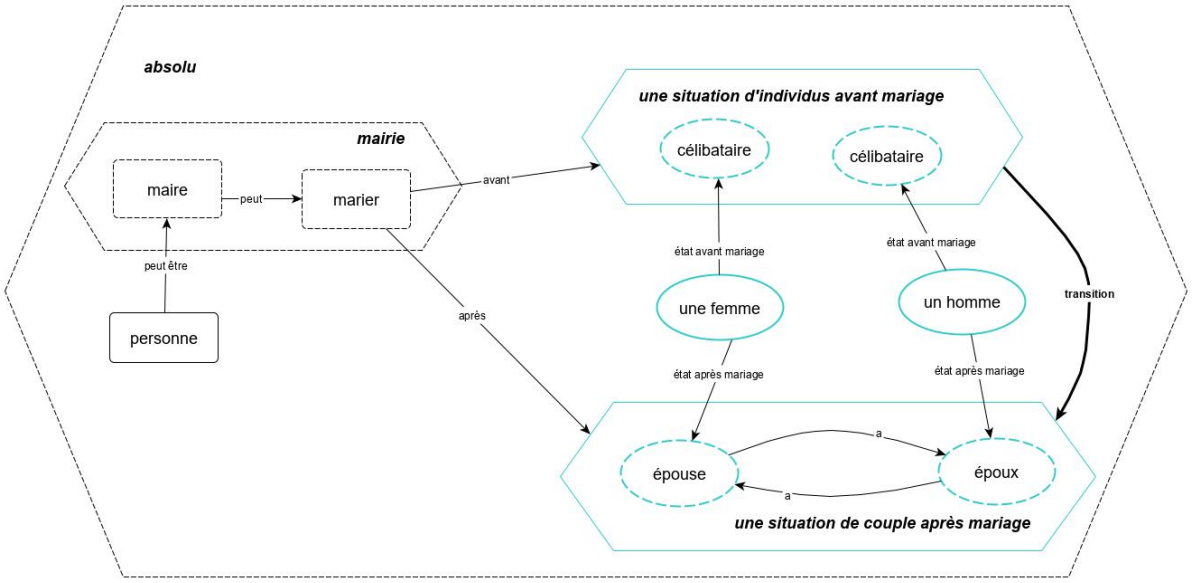
```
(((monde get: #faune) get: #Maya) getEtat: #miaulant) miaule
```

donne : Meow

Exemple 20 : sur les changements de situation

Pour cet exemple, nous allons repartir de l'exemple sur les relations entre manières d'être.

Le graphe suivant décrit le changement de situation d'une femme et d'un homme consécutif à leur mariage par une personne ayant la qualité de maire :



La représentation dans ICEO est la suivante :

```
1  iceo definition: #personne.
2  iceo definition: #femme genus: personne.
3  iceo definition: #homme genus: personne.
4  iceo definitionSituation: 'situation de couple'.
5  iceo definitionSituation: 'situation d\textquotesingle'individu'.
6  iceo definitionSituation: #mairie.
7  iceo definitionQualite: #épouse situation: (absolu getSituation: 'situation
       de couple').
8  iceo definitionQualite: #époux situation: (absolu getSituation: 'situation
      de couple').
9  iceo definitionQualite: #célibataire situation: (absolu getSituation: '
      situation d''individu').
10 iceo definitionQualite: #maire situation: (absolu getSituation: #mairie).
      iceo definitionQualiteEssentielle: #mariant pour: ((absolu getSituation:
       #mairie) get: #maire) effectivite: #intermittente.
11 ((absolu getSituation: 'situation de couple') get: #épouse)
      associationQualite: ((absolu getSituation: 'situation de couple')
            get: #époux).
12 femme peutEtre: ((absolu getSituation: 'situation de couple') get: #épouse)
      .
13 femme peutEtre: ((absolu getSituation: 'situation d''individu')
            get: #célibataire).
14 homme peutEtre: ((absolu getSituation: 'situation de couple') get: #époux).
15 homme peutEtre: ((absolu getSituation: 'situation d''individu')
            get: #célibataire).
16 personne peutEtre: ((absolu getSituation: #mairie) get: #maire).
```

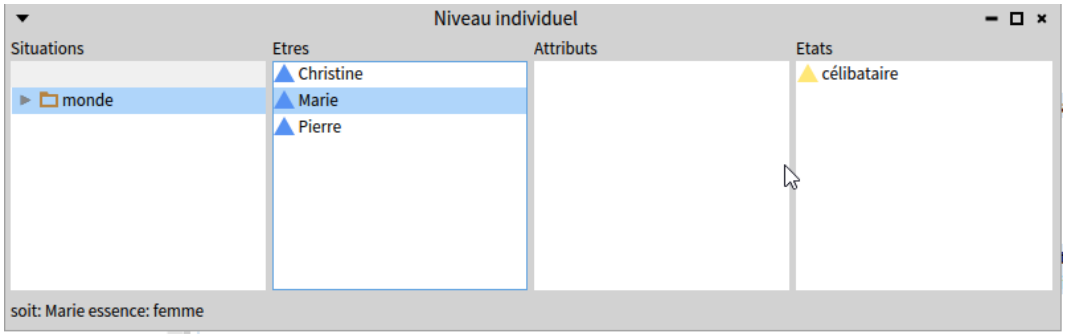
Supposons que la méthode suivante soit associée à la manière d'être mariant de la manière d'être maire :

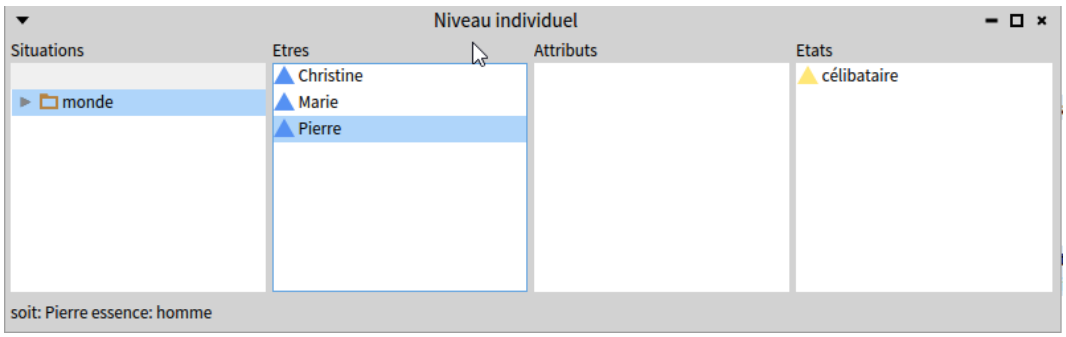
```
(((absolu getSituation: #mairie) get: #maire) getQualite: #mariant) compile: 'marie: s1 to: s2
        | unHomme uneFemme |
        s1 getElements
                do: [:etat |
                        etat getEssence getNom == #célibataire
                                ifTrue: [etat getEtant removeEtat: etat].
                        etat getEtant getEssence getNom == #homme
                                ifTrue: [unHomme := etat getEtant.
                                        unHomme
                                                affecteEtat: (s2 class get: #époux)
                                                dansSituation: s2]
                                ifFalse: [uneFemme := etat getEtant.
                                        uneFemme
                                                affecteEtat: (s2 class get: #épouse)
                                                dansSituation: s2]].
        (unHomme getEtat: #époux dansSituation: s2)
                associationEtat: (uneFemme getEtat: #épouse dansSituation: s2)'.
```

Christine, Marie et Pierre sont créés de la manière suivante :

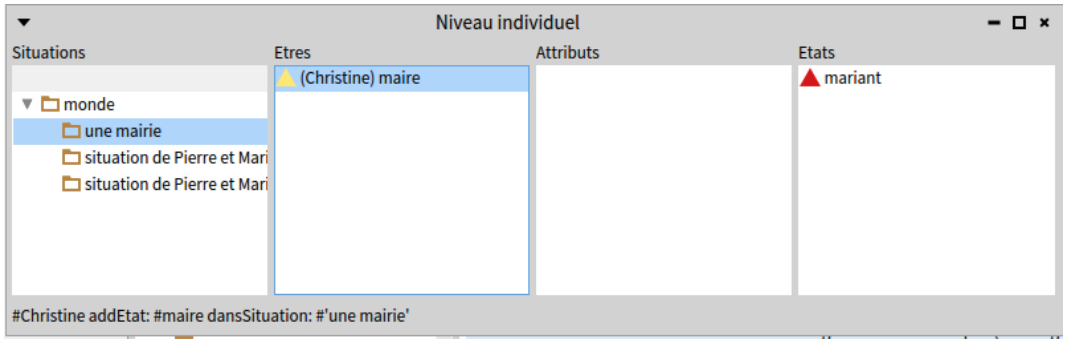
```
1  iceo soit: #Marie essence: femme.iceo soit: #Pierre essence: homme.
2  iceo soit: #Christine essence: personne.
3  iceo soit: 'une mairie' situationGenerique: (absolu getSituation: #mairie).
4  (monde get: #Christine) affecteEtat: ((absolu getSituation: #mairie) get: #
       maire) dansSituation: (monde getSituation: 'une mairie').
5  ((monde get: #Christine) getEtat: #maire) affecteEtatEssentiel: (((absolu
       getSituation: #mairie) get: #maire) getQualite: #mariant).
6  iceo soit: 'situation de Pierre et Marie avant mariage' situationGenerique:
        (absolu getSituation: 'situation d''individu').
7  iceo soit: 'situation de Pierre et Marie après mariage' situationGenerique:
        (absolu getSituation: 'situation de couple').
8  (monde get: #Marie) affecteEtat: ((absolu getSituation: 'situation d''
       individu\textquotesingle) get: #célibataire) dansSituation: (monde
       getSituation: 'situation de Pierre et Marie avant mariage').
9  (monde get: #Pierre) affecteEtat: ((absolu getSituation: 'situation d''
       individu') get: #célibataire) dansSituation: (monde getSituation: '
       situation de Pierre et Marie avant mariage').
10 personne peutEtre: ((absolu getSituation: #mairie) get: #maire).
```

Avant leur mariage, Pierre et Marie sont célibataires :



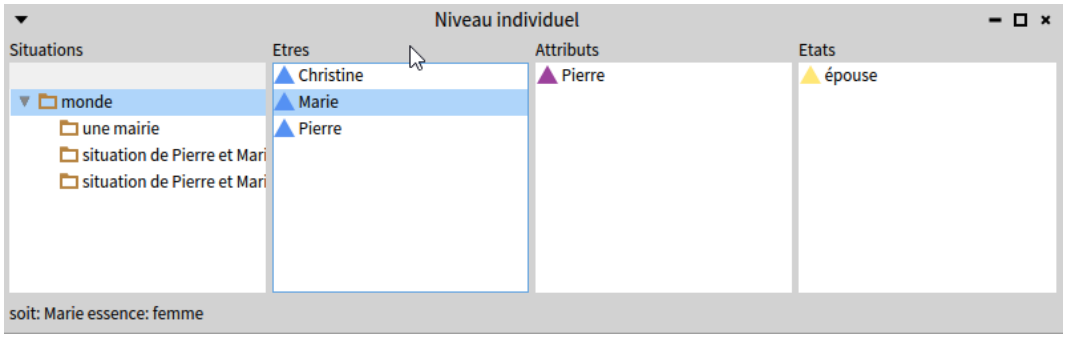


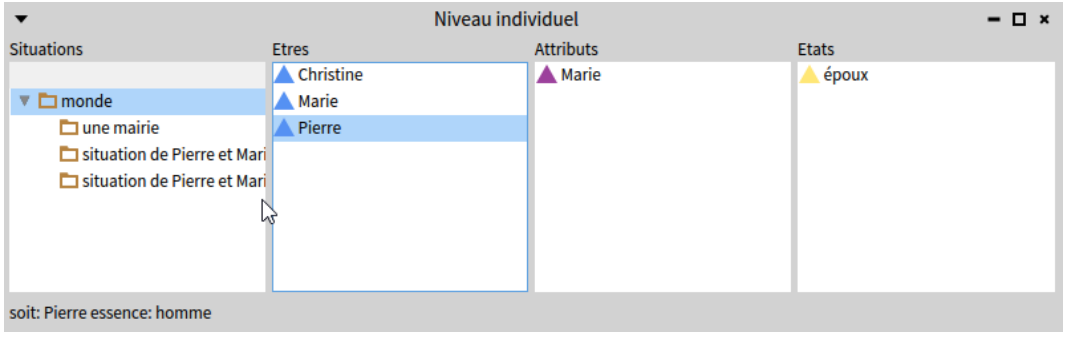
et Christine en tant que maire est dans l'état essentiel et intermittent mariant :



L'action de marier de Christine en tant que maire dans l'état mariant va les faire passer dans une situation de couple, avec les états d'épouse et d'époux :

```
(((monde get: #Christine) getEtat: #maire) getEtat: #mariant) marie:
(monde getSituation: 'situation de Pierre et Marie avant mariage') to:
(monde getSituation: 'situation de Pierre et Marie après mariage').
```





Après mariage on constate que Marie et Pierre ont bien été mariés par Christine en tant que maire.

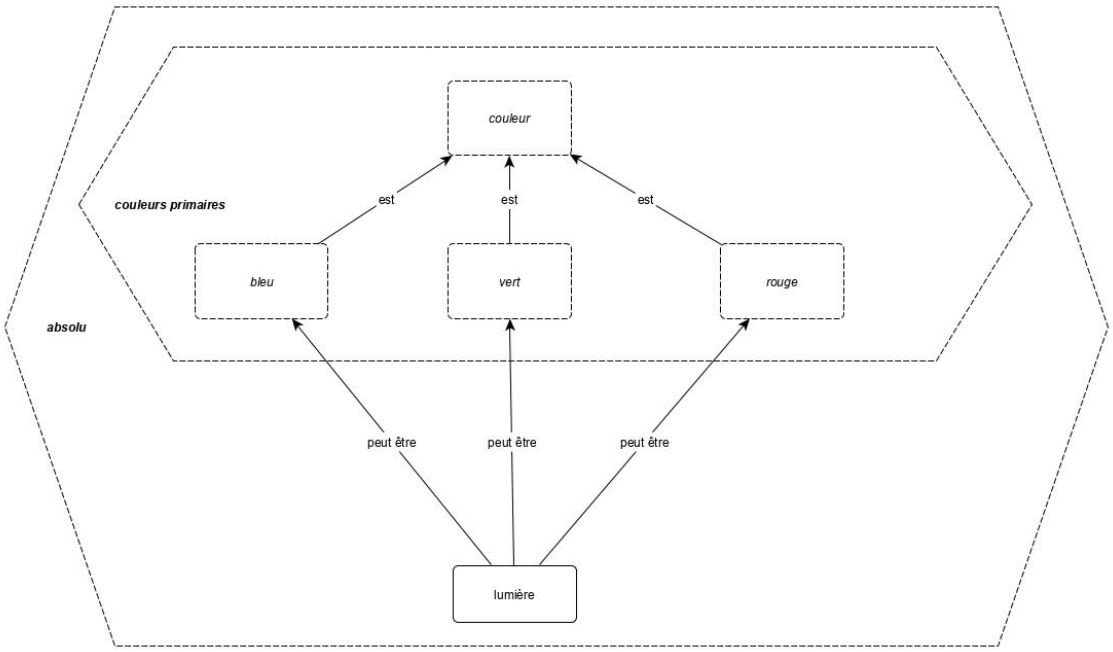
Exemple 21 : sur la notion de couleur

En soi, la lumière se présente comme de l'eau : elle n'a pas de forme, sa substance est massière.

Une lumière peut avoir différentes qualités. Ainsi une lumière peut être chaude (quand sa couleur tire vers l'orange) ou froide (lorsque sa couleur tire vers le bleu), . . . , de la même manière que l'eau peut nous paraître chaude, tiède, froide, . . .

Une couleur est une qualité de la lumière perçue par un sujet.

Voici la représentation des trois couleurs dites "primaires" :



Une lumière perçue peut être bleue, rouge ou verte.

Les couleurs primaires bleu, vert où rouge sont celles des trois (spectres de) lumières perçues et distinguées par l'œil humain que nous qualifierons de "monochromatiques"

Une lumière peut aussi être composée de lumières.

Les différentes compositions de lumières font apparaître d'autres couleurs possibles, reconnaissables par celui qui a appris à reconnaître ces compositions. Ainsi "blanc" est la couleur d'une lumière composée à parts égales de lumière rouge, verte et bleue.

Les couleurs des lumières composées à parts égales de deux lumières ayant une couleur primaire sont appelées secondaires. Ce sont les couleurs cyan, magenta et jaune.

Nous qualifierons ces lumières de "polychromatiques"

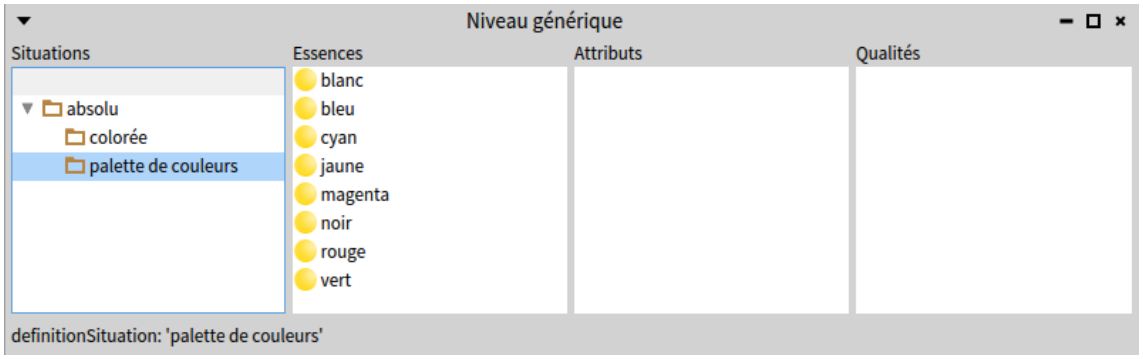
Les couleurs tertiaires viennent de la composition à parts égales d'une lumière ayant une couleur secondaire et de l'une des deux lumières de couleur primaire de sa couleur secondaire. Ainsi la couleur vermillon est composée à partir du jaune et du rouge.

noir est la couleur d'une lumière dont la composition est vide.

La palette des couleurs primaires et secondaires peut être définie de la manière suivante:

```
1  iceo definitionQualite: #couleur situation: (absolu getSituation: #
       colorée)
2  iceo definitionSituation: 'palette de couleurs'
3  iceo definitionQualite: #bleu situation: (absolu getSituation: '
       palette de couleurs') genus: ((absolu getSituation: #colorée)
       get: #couleur)
4  iceo definitionQualite: #vert situation: (absolu getSituation: '
       palette de couleurs') genus: ((absolu getSituation: #colorée)
       get: #couleur)
5  iceo definitionQualite: #rouge situation: (absolu getSituation: '
       palette de couleurs') genus: ((absolu getSituation: #colorée)
       get: #couleur)
6  iceo definitionQualite: #cyan situation: (absolu getSituation: '
       palette de couleurs') genus: ((absolu getSituation: #colorée)
       get: #couleur)
7  iceo definitionQualite: #magenta situation: (absolu getSituation: '
       palette de couleurs') genus: ((absolu getSituation: #colorée)
       get: #couleur)
8  iceo definitionQualite: #jaune situation: (absolu getSituation '
       palette de couleurs') genus: ((absolu getSituation: #colorée)
       get: #couleur)
9  iceo definitionQualite: #blanc situation: (absolu getSituation: '
       palette de couleurs') genus: ((absolu getSituation: #colorée)
       get: #couleur)
10 iceo definitionQualite: #noir situation: (absolu getSituation '
       palette de couleurs') genus: ((absolu getSituation: #colorée)
       get: #couleur)
```

La palette de couleurs ainsi définie est :

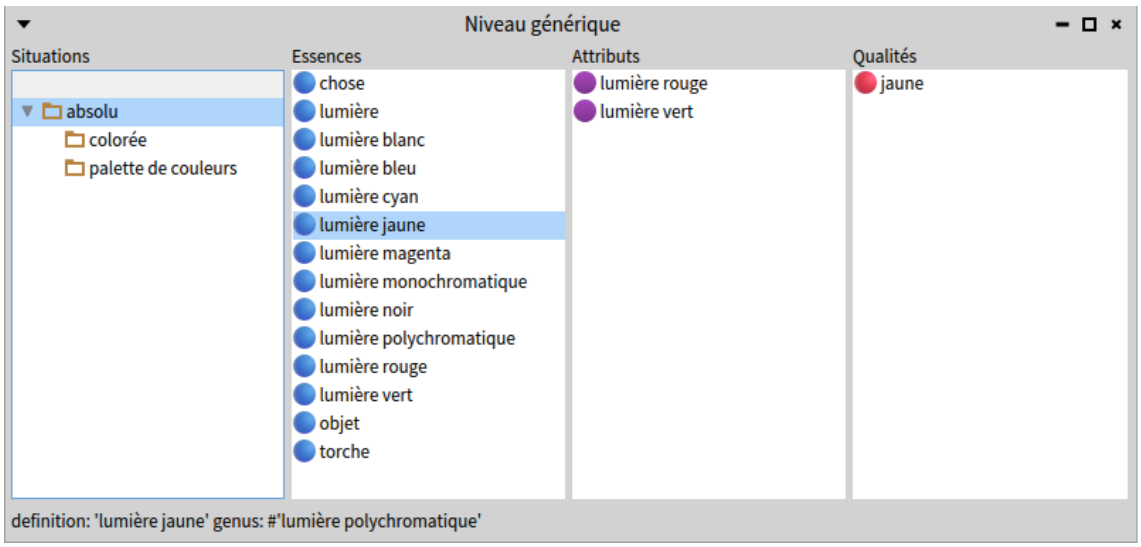


Voici comment peuvent se définir les différentes lumières :

```
1  iceo definition: #lumière.
2  iceo definition: 'lumière monochromatique' genus: lumière.iceo
       definition: 'lumière polychromatique' genus: lumière.
3  iceo definition: 'lumière bleu' genus: (absolu get: 'lumiè
       remonochromatique').
4  iceo definitionQualiteEssentielle: 'bleu' pour: (absolu get: 'lumiè
       re bleu') effectivite: #permanente.
```

```
5         iceo definition: 'lumière vert' genus: (absolu get: 'lumière
              monochromatique').
6         iceo definitionQualiteEssentielle: 'vert' pour: (absolu get: 'lumiè
              re vert') effectivite: #permanente.
7         iceo definition: 'lumière rouge' genus: (absolu get: 'lumière
              monochromatique').
8         iceo definitionQualiteEssentielle: 'rouge' pour: (absolu get: 'lumi
              ère rouge') effectivite: #permanente.
9         iceo definition: 'lumière cyan' genus: (absolu get: 'lumière
              polychromatique').
10        iceo definitionQualiteEssentielle: 'cyan' pour: (absolu get: 'lumiè
              re cyan') effectivite: #permanente.
11        (absolu get: 'lumière cyan\textquotesingle) referenceEssence: (
              absolu get: 'lumière bleu') cardinalite: 1.
12        (absolu get: 'lumière cyan\textquotesingle) referenceEssence: (
              absolu get: 'lumière vert') cardinalite: 1.
13        iceo definition: 'lumière magenta' genus: (absolu get: 'lumière
              polychromatique').
14        iceo definitionQualiteEssentielle: 'magenta' pour: (absolu get: '
              lumière magenta') effectivite: #permanente.
15        (absolu get: 'lumière magenta\textquotesingle) referenceEssence: (
              absolu get: 'lumière bleu') cardinalite: 1.
16        (absolu get: 'lumière magenta\textquotesingle) referenceEssence: (
              absolu get: 'lumière rouge') cardinalite: 1.
17        iceo definition: 'lumière jaune' genus: (absolu get: 'lumière
              polychromatique').
18        iceo definitionQualiteEssentielle: 'jaune' pour: (absolu get: 'lumi
              ère jaune') effectivite: #permanente.
19        (absolu get: 'lumière jaune\textquotesingle) referenceEssence: (
              absolu get: 'lumière vert') cardinalite: 1.
20        (absolu get: 'lumière jaune\textquotesingle) referenceEssence: (
              absolu get: 'lumière rouge') cardinalite: 1.
21        iceo definition: 'lumière blanc' genus: (absolu get: 'lumière
              polychromatique').iceo definitionQualiteEssentielle: 'blanc'
              pour: (absolu get: 'lumière blanc') effectivite: #permanente.
22        (absolu get: 'lumière blanc') referenceEssence: (absolu get: 'lumiè
              re vert') cardinalite: 1.
23        (absolu get: 'lumière blanc') referenceEssence: (absolu get: 'lumiè
              re rouge') cardinalite: 1.
24        (absolu get: 'lumière blanc') referenceEssence: (absolu get: 'lumiè
              re bleu') cardinalite: 1.
25        iceo definition: 'lumière noir' genus: (absolu get: 'lumière
              polychromatique').
26        iceo definitionQualiteEssentielle: 'noir' pour: (absolu get: 'lumiè
              re noir') effectivite: #permanente.
```

L'ensemble de lumières ainsi défini est :



On voit que par exemple la lumière jaune possède la qualité essentielle d'être jaune, et qu'elle est composée des lumières rouge et vert.

Couleur des objets

La couleur d'un objet est celle de la lumière qu'il reflète ou diffuse lorsqu'il est éclairé.

un objet éclairé avec une lumière peut être coloré comme il peut être mouillé avec un liquide.

Les capacités de diffusion de la lumière d'un objet sont liées à sa composition physique; elles ne dépendent pas de l'éclairage mais déterminent sa couleur une fois éclairé. Elles sont bien sûr les mêmes en l'absence de lumière (dans le noir).

En peinture, la composition des couleurs est basée sur les capacités de diffusion de la lumière par la peinture. La composition est soustractive. Les trois couleurs fondamentales en peinture sont cyan, magenta et jaune. La peinture verte est obtenue par mélange de pigments jaune et cyan :

Par convention, la couleur d'un objet est celle qu'il possède éclairé par de la lumière blanche. C'est sa couleur prise conventionnellement par défaut, que nous qualifierons de couleur "objective". Avec cette convention, éclairé par de la lumière rouge, un objet jaune paraîtra rouge (car la couleur objective jaune signifie que l'objet diffuse les lumières rouge et vert).

Quel que soit l'éclairage, un objet noir apparaitra toujours noir, car il absorbe toutes les lumières.

Nous qualifierons de "couleur apparente" la couleur que prend un objet avec un éclairage particulier.

Autrement dit, c'est le couple (couleur objective, éclairage) qui détermine la couleur apparente de l'objet. Changer l'éclairage entraîne un changement de couleur apparente

des objets éclairés. En l'absence d'éclairage, c'est-à-dire dans le noir, un objet parait noir.

L'éclairage est le contexte où la couleur d'un objet devient perceptible.

Voici la définition d'un objet de couleur jaune :

```
1        iceo definition: #objet.objet peutEtre: ((absolu getSituation: #
             colorée) get: #couleur).
2        iceo soit: 'un objet' essence: objet.
3        (monde get: 'un objet') affecteEtat: ((absolu getSituation: '
             palette de couleurs') getElement: #jaune) dansSituation: (monde
             get: 'une palette de couleurs').
```

Et voici la définition d'une torche émettant une lumière magenta :

```
1        iceo definition: #torche.torche referenceEssence: lumière.
2        iceo soit: 'une torche' essence: torche.
3        (monde get: 'une torche') attributionEtre: 'une lumière' essence: (
             absolu get: 'lumière magenta').
```

Le code Smalltalk suivant définit une méthode de l'essence objet qui infère la couleur apparente d'un objet éclairé par une torche, en faisant l'intersection de l'ensemble de lumières émises par la torche avec l'ensemble des lumières reflétées par l'objet :

```
objet compile: 'couleurApparenteEclairePar: uneTorche
| l c s t |
s := (absolu getElements: lumière) select: [:each |
 (each getGenus == (absolu get: "lumière monochromatique")) or: [each getGenus == (absolu get: "lumière
polychromatique")]].
t := s detect: [:each |
 l := each.
 l getGenus == (absolu get: "lumière monochromatique")

ifTrue: [c := OrderedCollection with: l ]
ifFalse: [ c := l getEssencesAttributs sort: [:a :b | a getNom  < b getNom] ] .
c  = ((uneTorche lumièresEmises  intersection: self lumièresReflétées) sort: [:a :b | a getNom  < b getNom]) ] .
        ^t getQualites at: 1  '.
```

Cette méthode utilise deux autres méthodes qui déterminent respectivement les lumières reflétées par un objet et les lumières émises par une torche :

```
objet compile: 'lumièresReflétées
        | l |
        l := (absolu get: "lumière ", (self  getEtatEssence: ((absolu getSituation: #colorée) get: #couleur)) getNom).
        l getGenus == (absolu get: "lumière monochromatique") ifTrue: [^ OrderedCollection with: l ]
        ifFalse: [ ^ l getEssencesAttributs sort: [:a :b | a getNom  < b getNom] ]  '.

torche  compile: 'lumièresEmises
        | l |
        l := (self getEtreAttribut: #lumière) getEssence.
        l getGenus == (absolu get: "lumière monochromatique") ifTrue: [^ OrderedCollection with: l ]
        ifFalse: [ ^ l  getEssencesAttributs  sort: [:a :b | a getNom  < b getNom]  ] '.
```

Ainsi par exemple, pour un objet de couleur jaune éclairé par de la lumière magenta, l'expression :

```
(monde get: 'un objet') couleurApparenteEclairePar: (monde get: 'unetorche')
dans: monde
```

retourne : rouge

Pour un objet de couleur jaune éclairé par de la lumière bleue, la couleur apparente serait le noir, ...

Exemple 22 : les tours de Hanoi revisitées

La solution que nous proposons ici est inspirée de celle proposée par Ted Kaehler et Dave Patterson dans le livre "A TASTE OF SMALLTALK".

Comme les auteurs de ce livre le font remarquer, personne ne va résoudre le problème des tours de Hanoi suivant un algorithme récursif qui est obscur. Si on stoppe l'algorithme au milieu, où est stocké l'état de chaque disque ?

Les auteurs ont proposé une solution où les disques sont des objets qui se déplacent en fonction des messages déterminés par un chef d'orchestre. Aux deux règles du jeu est ajoutée une troisième (qui semble logique pour avoir un minimum de déplacements) pour interdir qu'un disque ne revienne en arrière.

Notre solution diffère sur différents points :

- les disques sont des acteurs (implantés sous forme de processes) qui jouent une course de relais.

- au lieu de placer des "mock disks" plus grands que tous les autres sur les piquets vides, nous avons défini les manières d'être relatives "sous" et "sur" pour les disques.

```
1 iceo definition: #jeu.iceo definitionSituation: #piquet.
2 jeu referenceEssence: piquet cardinalite: 3.
3 iceo definitionAttribut: #disque de: jeu cardinalite: 6.
4 iceo definitionQualite: #sous situation: absolu.
5 (absolu get: #sous) referenceEssence: (jeu getEssenceAttribut: #disque).
6 (jeu getEssenceAttribut: #disque) peutEtre: (absolu get: #sous).
7 iceo definitionQualite: #sur situation: absolu.
8 (absolu get: #sur) referenceEssence: (jeu getEssenceAttribut: #disque).
9 (jeu getEssenceAttribut: #disque) peutEtre: (absolu get: #sur).
```

Les piquets, au nombre de trois, sont les situations où évoluent les disques.

Au départ, les disques sont empilés sur le premier piquet en tenant compte de leur taille :

```
1 iceo soit: #Hanoi essence: jeu.
2 " situation intiale des disques sur le piquet piquet\_0"
3 ((monde get: #Hanoi) getEtresAttributs: #disque) do: {[]:d \textbar{} ((
     monde get: #Hanoi) getEtreAttribut: #piquet\_0) introductionEtre: d{]}.
4 " état intial relatif des disques "
5 (Interval from: 1 to: (((monde get: #Hanoi) getEtresAttributs: #disque)
     size -1)) do: {[]:i \textbar{} (((monde get: #Hanoi) getEtresAttributs:
     #disque) at: i) affecteEtat: (absolu get: #sous) dansSituation: monde.
6 ((((monde get: #Hanoi) getEtresAttributs: #disque) at: i) getEtat: #sous)
     attributionEtre: (((monde get: #Hanoi) getEtresAttributs: #disque) at: i
     +1).
7 (((monde get: #Hanoi) getEtresAttributs: #disque) at: i+1) affecteEtat: (
     absolu get: #sur) dansSituation: monde.
8 ((((monde get: #Hanoi) getEtresAttributs: #disque) at: i+1) getEtat: #sur)
     attributionEtre: (((monde get: #Hanoi) getEtresAttributs: #disque) at: i
     ){]}.
```

Chaque disque possède un comportement qui est une manière d'être essentielle et permanente.

```
iceo definitionQualiteEssentielle: #comportement pour: (jeu
    getEssenceAttribut: #disque) effectivite: #permanente.
```

Au comportement de chaque disque est associé un process Smalltalk et un sémaphore[1] :

```
" déclaration des joueurs "
        (monde get: #Hanoi) getEtres do: [ :each |
                | choix started |
                started := false.
                (each getEtat: #comportement) setSemaphore: Semaphore new.
                (each getEtat: #comportement) setProcess: ([ :h :d |
                        | s |
                        s isNil ifTrue: [ s := false ].
                        [ true ] whileTrue: [
                                s ifFalse: [
                                        (d getNom , ' is ready') crTrace.
                                        s := true.
                                        (d getEtat: #comportement) getSemaphore wait ].
                                " qui peut se déplacer ? "
                                choix := h getEtres detect: [ :disque | disque peutSeDeplacer ].
                                choix == d
                                        ifTrue: [ "c'est moi"
                                                d sprint.
                                                "qui peut maintenant se déplacer en dehors de moi ? "
                                                h getEtres
                                                        detect: [ :disque | disque peutSeDeplacer ]
                                                        ifFound: [ :x |
                                                        (x getEtat: #comportement) getSemaphore signal ].
                                                (d getEtat: #comportement) getSemaphore wait ]
                                        ifFalse: [ "ce n'est pas moi"
                                                (choix getEtat: #comportement) getSemaphore signal.
                                                (d getNom , ' is waiting') crTrace.
                                                ' ' crTrace.
                                                (d getEtat: #comportement) getSemaphore wait ] ] ]
                        newProcessWith: (Array with: (monde get: #Hanoi) with: each)) ].
```

Au départ, les joueurs sont placés dans un état d'attente par la méthode de jeu "allezAuxStartingBlocks".

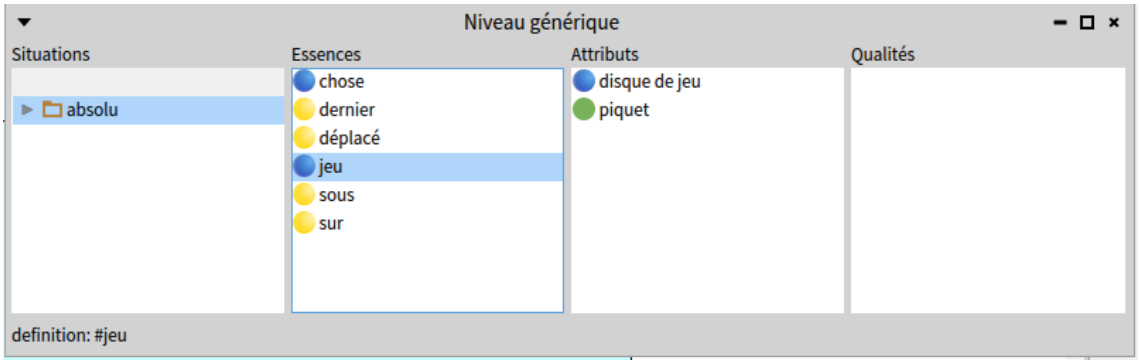
Pour lancer le jeu, sélectionner l'ensemble des lignes de l'exemple "hanoi.text" et faire un "Do it"

La fenêtre suvante s'affiche :

[1]Les notions de process et de sémaphore en Smalltalk sont décrits dans le livre "Concurrent Programming in Pharo" de Stéphane Ducasse et Guillermo Polito".

On voit que le nombre de disques a été ici fixé à 6. Ce nombre peut être changé dans le code de l'exemple. Pour que les disques tiennent dans la fenêtre, éviter de prendre un nombre supérieur à 7. Noter que si vous fixez ce nombre à 64, le nombre de déplacements possibles sera de 2^{64}-1, soit 18 446 744 073 709 551 615. Il faudra être patient !.

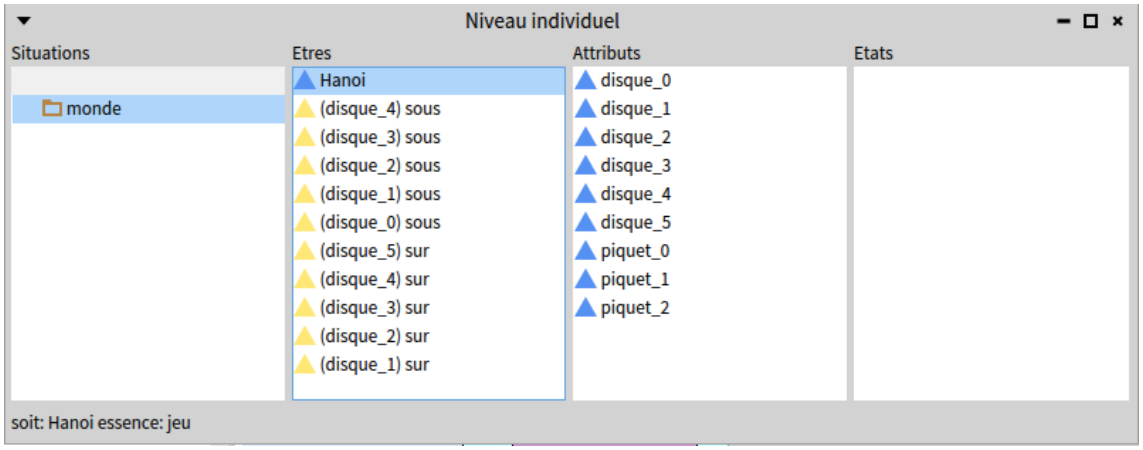
Il peut être intéressant à ce stade d'ouvrir un SgBrowser :



On voit que l'essence jeu possède deux attributs, "disque de jeu" et "piquet". Le petit cercle vert devant l'attribut piquet indique qu'il s'agit d'une situation générique.

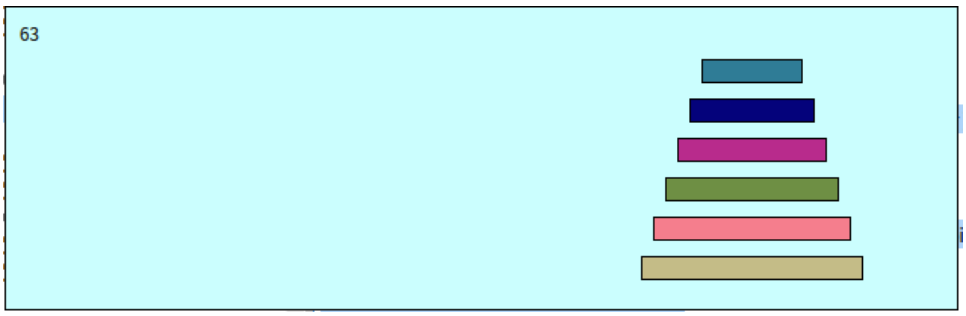
Chaque piquet est effectivement une situation où évoluent les disques. Noter que dans ICEO rien n'empêche qu'une situation soit attribut d'un être.

Au niveau individuel, on voit que l'instance de jeu nommée "Hanoi" possède trois piquets et 6 disques. On peut vérifier que tous les disques sauf le disque 6 sont dans l'état "sous" et que tous les disques sauf le disque 0 sont dans l'état "sur".



Un click avec le bouton gauche de la souris dans la fenêtre où sont visualisés les disques envoie un signal au plus petit pour lui dire de commencer la course.

Vous voyez les disques se déplacer dans une course de relais qui se termine lorsque les disques ont retouvé leur configuration initiale, sur un autre piquet.



Par défaut, un intervalle de temps (modifiable) entre les déplacements a été fixé à 100 ms. Le nombre affiché en haut à gauche de la fenêtre correspond au nombre de déplacements effectués par les disques. Le nombre 63 correspond bien à la valeur 2^6-1 qui est la valeur théorique minimale pour ce jeu avec six disques.

Pour clore le jeu, il suffit d'un click avec le bouton droit de la souris dans la fenêtre.

En cours de jeu, une fenêtre de type Transcript affiche les déplacements effectués :

```
Transcript
disque_5 : piquet_1 ---> piquet_2
disque_3 : piquet_0 ---> piquet_1
disque_5 : piquet_2 ---> piquet_0
disque_4 : piquet_2 ---> piquet_1
disque_5 : piquet_0 ---> piquet_1
disque_2 : piquet_0 ---> piquet_2
disque_5 : piquet_1 ---> piquet_2
disque_4 : piquet_1 ---> piquet_0
disque_5 : piquet_2 ---> piquet_0
disque_3 : piquet_1 ---> piquet_2
disque_5 : piquet_0 ---> piquet_1
disque_4 : piquet_0 ---> piquet_2
disque_5 : piquet_1 ---> piquet_2
See you soon !
```

Sur l'implantation d'ICEO en Pharo

Il y a peu de choses à dire sur cette implantation, tant ICEO s'intègre naturellement dans le langage Smalltalk. Le code source d'ICEO comporte 2700 lignes en incluant les commentaires.

Aucune méthode de l'environnement de base de Pharo n'a été modifiée, à part les méthodes "evaluateSelection" de RubSmalltalkEditor et "evaluate: aString onCompileError: compileErrorBlock onError: errorBlock" de SpCodePresenter, pour éviter que le compilateur n'évalue une ligne avant d'avoir terminé d'évaluer la précédente ! (ce qui permet d'évaluer toutes les lignes de chaque exemple en bloc).

La méthode "validateClassName" de ShiftClassBuilder a également été modifiée pour autoriser que le nom des classes ne commencent pas par une majuscule. Pour permettre l'homonymie des essences, nous avons mis à profit la possibilité de créer des classes de type "newAnonymousSubclass".

La classe Essence a comme sous-classes directes les essences "chose" (racine de la hiérarchie des essences dans ICEO) et "absolu" qui est la racine des situations génériques.

Les attributs des essences sont des instances de la classe Association, ce qui permet d'associer à chaque essence attribut une cardinalité pour sa composition.

Pour la composition des essences qui sont des manières d'être et des situations génériques, celles-ci sont traitées comme des essences (avec leur differentia), à la différence près que leurs attributs n'ont pas de cardinalité.

La classe ICEO correspond à l'interprète d'ICEO, utilisée pour la définition des situations, des essences et de leurs qualité et l'instanciation des êtres.

La classe Etre est superclasse de la classe Essence. Les attributs et les méthodes de classe de la classe Etre sont les mêmes que ceux des instances de la classe Essence, ce qui permet de voir toute essence comme un être instance de sa propre essence (sa méta-essence).

Par défaut, la méta-essence d'une essence est l'essence chose et l'essence chose est son propre genus et sa propre méta-essence.

API d'ICEO

Classe ICEO, interprète du langage ICEO

Méthodes d'instance

Méthodes concernant les qualités

definitionQualite: unStringOrSymbol situation: uneSituation
définition dans uneSituation d'une manière d'être subsumée par chose

definitionQualiteEssentielle: unStringOrSymbol pour: uneEssence effectivite: unSymbole
définition dans la situation de définition de uneEssence d'une manière d'être essentielle subsumée par chose

definitionQualiteEssentielle: unStringOrSymbol genus: uneEssence pour: uneAutreEssence effectivite: unSymbole
définition dans la situation de définition de uneAutreEssence d'une manière d'être essentielle subsumée par uneEssence

definitionQualite: unStringOrSymbol situation: uneSituation genus: uneEssence
définition dans uneSituation d'une manière d'être subsumée par uneEssence

Méthodes concernant les êtres et les états

soitAttribut: unStringOrSymbol de: unEtre essence: uneEssence
définition d'un attribut d'un être d'essence uneEssence

soit: unStringOrSymbol essence: uneEssence situationIndividuelle: uneSituation quiEst: uneQualite
définition dans uneSituationIndividuelle d'un être d'essence uneEssence ayant un état instance de uneQualité

soitEtat: unStringOrSymbol essence: uneQualite situationIndividuelle: uneSituation
définition dans uneSituationIndividuelle d'un état d'essence uneQualite dont l'étant est indéterminé

soit: unStringOrSymbol essence: uneEssence situationIndividuelle: uneSitua-

tion
définition dans uneSituationIndividuelle d'un être d'essence uneEssence

soit: unStringOrSymbol essence: uneEssence
définition dans le monde d'un être d'essence uneEssence

soit: unStringOrSymbol essence: uneEssence
définition dans le monde d'un être d'essence uneEssence

Méthodes concernant les situations

soit: unStringOrSymbol situationGenerique: uneSituationGenerique dansSituationIndividuelle: uneSituationIndividuelle
définition dans uneSituationIndividuelle d'une situation individuelle instance de la situation générique uneSituation

soit: unStringOrSymbol situationGenerique: uneSituation
définition dans le monde d'une situation individuelle instance de la situation générique uneSituation définie dans l'absolu

definitionSituation: unStringOrSymbol dans: uneSituationGenerique
définition dans uneSituationGenerique d'une situation générique subsumée par Essence

definitionSituation: unStringOrSymbol
définition dans absolu d'une situation générique subsumée par Essence

Méthodes concernant les essences

definitionAttribut: unStringOrSymbol de: uneEssence cardinalite: uneValeur
définition dans le differentia de une essence d'une essence subsumée par chose

definitionAttribut: unStringOrSymbol de: uneEssence genus: uneAutreEssence
définition dans le differentia de une essence d'une essence subsumée par uneAutreEssence

definition: unStringOrSymbol situation: uneSituation genus: uneEssence
définition dans uneSituation d'une essence subsumée par uneEssence

definition: unStringOrSymbol
définition dans absolu d'une essence subsumée par chose

definition: unStringOrSymbol genus: uneEssence metaEssence: uneAutreEssence
définition dans l'absolu d'une essence subsumée par uneEssence et instance de uneAutreEssence

definitionAttribut: unStringOrSymbol de: uneEssence isPrototype: aBoolean
définition dans le differentia de une essence d'une essence subsumée par chose

definitionAttribut: unStringOrSymbol de: uneEssence genus: uneAutreEssence cardinalite: uneValeur

définition dans le differentia de une essence d'une essence subsumée par uneAutreEssence

definitionAttribut: unStringOrSymbol de: uneEssence isPrototype: aBoolean cardinalite: uneValeur
définition dans le differentia de une essence d'une essence subsumée par chose qui est prototype

definitionAttribut: unStringOrSymbol de: uneEssence
définition dans le differentia de une essence d'une essence subsumée par chose

definition: unStringOrSymbol genus: uneEssence
définition dans absolu d'une essence subsumée par uneEssence

definition: unStringOrSymbol situation: uneSituation
définition dans uneSituation d'une essence subsumée par chose

definition: unStringOrSymbol situation: uneSituation
définition dans uneSituation d'une essence subsumée par chose

Méthodes de classe

forDoItInPlayground
Permet que les lignes évaluées en bloc soient évaluées en séquence dans une fenêtre de type Playground et attrape les exceptions de classe Oups

forDoIt
Permet que les lignes évaluées en bloc soient évaluées en séquence dans une fenêtre de type FileBrowser et attrape les exceptions de classe Oups

genSym: aSymb
generateur de symboles suffixés par des nombres définis en séquence

getPremierGenusCommun: uneCollectionEssences
détermine le premier genus commun à deux essences

ajoute: aBuilder
ajoute un onglet ICEO dans la fenêtre principale de Pharo

addSgBrowser: aBuilder
ajoute un item sgBrowser dans l'onglet ICEO de la fenêtre principale de Pharo

addSiBrowser: aBuilder
ajoute un item siBrowser dans l'onglet ICEO de la fenêtre principale de Pharo

addReset: aBuilder
ajoute un item Reset dans l'onglet ICEO de la fenêtre principale de Pharo

start
Appelée lors du chargement de ICEO.st, en particulier pour créer l'essence chose et les situations absolu et monde

reset

supprime toutes les essences subsumées par chose et les situations incluses dans absolu et monde

buildIcons

crée toutes les icônes utilisées dans les browsers

Classe Essence, superclasse de l'essence chose et de la situation absolu

Méthodes d'instance

instanceVariableNames

A toutes ces variables sont associées des méthodes get et set :
structure situationDefinition isSituation isEtat etats etant nom id description

Méthodes concernant les états

removeEtat: unEtat
supprime l'état unEtat d'un être (un étant)

removeEtatDeNom: unStringOrSymbol
supprime un état d'un être (un étant)

affecteEtatEssentiel: uneQualite
affecte à un être un état instance d'une manière d'être essentielle

getEtat: unStringOrSymbol dansSituation: uneSituation
retourne un état situé dans uneSituation

getEtats: unStringOrSymbol dansSituation: uneSituation
retourne les états de nom sunStringOrSymbol situés dans uneSituation

getEtatEssence: uneEssence
retourne les états instances d'une manière d'être uneEssence

associationEtat: unEtat
crée une association bidirectionnellle entre self (un état) et unEtat

affecteEtat: uneQualite dansSituation: uneSituationIndividuelle
affecte à un être un état instance d'une manière d'être uneQualite dans uneSituationIndividuelle

Méthodes concernant les situations individuelles

getSituation: unSymbole
retourne les situations individuelles de nom unSymbole incluses dans self (une situation individuelle

getElements
retourne les êtres situés dans self (une situation individuelle)

getSituationGenerique
retourne la situation générique de self (une situation individuelle)

getSituations
retourne les situations individuelles incluses dans self (une situation individuelle)

getElementsEssence: uneEssence
pour une situation, retourne l'ensemble des êtres ayant uneEssence pour essence)

introductionEtre: unEtre
dans une situation individuelle

get: unStringOrSymbol
pour une situation individuelle, retourne, s'il existe, l'etre nommé unStringOrSymbol

get: unStringOrSymbol quiEst: uneQualite
pour une situation individuelle, retourne, s'il existe, l'etre nommé unStringOrSymbol qui est dans un état instance de la manière d'être uneQualite

Méthodes concernant les êtres

attributionEtre: unEtre
change la situation de définition de unEtre

attributionEtre: unStringOrSymbol essence: uneEssence
attribue un nouvel être nommé unStringOrSymbol d'essence uneEssence

getEtresAttributsQuiSont: uneQualite
retourne les êtres attributs qui dans un état instance de la manière d'être uneQualite

getEtreAttribut: unStringOrSymbol
retourne l'être attribut nommé unStringOrSymbol

getEtresAttributs
retourne les êtres attributs propres et "hérités" s'ils sont définis comme prototypes au niveau méta

getAllEtresAttributs
retourne les êtres attributs propres et "acquis" via les états

getEtresAttributsEnTantQue: uneQualite
retourne les êtres attributs "acquis" en étant dans un état instance de la manière d'être uneQualite

getEtresAttributs: unStringOrSymbol
retourne les êtres attributs de nom unStringOrSymbol

getEtresAttributs: unStringOrSymbol
retourne les êtres attributs de nom unStringOrSymbol

Méthodes de classe

instanceVariableNames

A toutes ces variables sont associées des méthodes get et set :
differentia isQualite isIndividu isPrototype statut effectivite qualites essencesQualifiees metaEssence

Méthodes concernant les essences

getEssencesAttributs
retourne l'ensemble des attributs (essences) propres et hérités et, pour une manière d'être, les qualites associées

getEssencesAttributs: unStringOrSymbol
retourne l'ensemble des attributs (essences) propres et hérités et acquis par des manières d'être de nom unStringOrSymbol

getEssenceAttribut: unStringOrSymbol
retourne l'attribut (essence) propre ou hérités ou, pour une manière d'être, la qualite associée, de nom unStringOrSymbol

getAttributs
retourne l'ensemble des attributs (instances d'Association liant une essence à une cardinalité) propres et hérités mais non surchargés pour une essence

getAttribut: unStringOrSymbol
retourne l'attribut (instance d'Association liant une essence à une cardinalité) propre ou hérité mais non surchargé pour une essence, de nom unStringOrSymbol

getEssence
par défaut, l'essence d'une essence est l'essence chose

getGenus
retourne l'essence subsumante

getAllGenus
retourne l'ensemble des essences subsumantes

getAllGenusWithoutMe
retourne l'ensemble des essences subsumantes en excluant self

subsume: uneEssence
retourne true si uneEssence subsume self

specialise: uneEssence
retourne true si self subsume unEssence

isChose
retourne true si self est subsumée par chose

referenceEssence: uneEssence cardinalite: anInteger

"introduit uneEssence dans le differentia sans changer la situation de définition de uneEssence"

referenceEssence: uneEssence

"identique à referenceEssence: uneEssence cardinalite: anInteger avec une cardinalité non définie"

Méthodes concernant les manières d'être

peutEtre: uneQualite

affecte une manière d'être possible pour une essence

est: uneQualite

affecte une manière d'être essentielle pour une essence

getQualitesPropres

retourne l'ensemble des manières d'êtres d'une essence

getQualitesAssociees

retourne l'ensemble des manières d'êtres associées à une manière d'être

getEssencesQualifiees

retourne l'ensemble des essences qualifiées par une manière d'être

associationQualite: uneQualite

crée une association binaire et bidirectionnelle entre une manière d'être et une autre (uneQualite)

getQualite: unStringOrSymbol

retourne la manière d'être nommée unStringOrSymbol incluse dans les qualités d'une essence

getQualites

retourne l'ensemble des manières d'être propres et héritées mais non surchargées pour une essence

Méthodes concernant les situations génériques

getEssences

retourne les essences incluses dans une situation générique

getElements

retourne l'ensemble des éléments inclus dans une situation générique

getElements: uneEssence

retourne l'ensemble des éléments inclus dans une situation générique qui sont subsumés par uneEssence

getElement: unStringOrSymbol

retourne l'essence unStringOrSymbol incluse dans une situation

get: unStringOrSymbol
identique à getElement: unStringOrSymbol

getSituationsGeneriques
retourne l'ensemble des situations génériques incluse dans une situation génériques

Méthodes diverses

getInstance: unStringOrSymbol
création d'un être hypothétique instance de self

createFreeSubclass: aSymbol
création d'une sous-classe qui n'est pas installée dans le système de classes de Smalltalk